



**NAVAL
POSTGRADUATE
SCHOOL**

MONTEREY, CALIFORNIA

THESIS

**ANALYSIS AND DEVELOPMENT OF A WEB-ENABLED
PLANNING AND SCHEDULING DATABASE APPLICATION**

by

Gary L. Reed

September 2013

Thesis Advisor: Glenn R. Cook
Second Reader: William J. Robinette

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE		Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE September 2013	3. REPORT TYPE AND DATES COVERED Master's Thesis
4. TITLE AND SUBTITLE ANALYSIS AND DEVELOPMENT OF A WEB-ENABLED PLANNING AND SCHEDULING DATABASE APPLICATION		5. FUNDING NUMBERS	
6. AUTHOR(S) Gary L. Reed		8. PERFORMING ORGANIZATION REPORT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000		10. SPONSORING/MONITORING AGENCY REPORT NUMBER	
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A		11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. government. IRB protocol number _____ N/A _____.	
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited		12b. DISTRIBUTION CODE A	
13. ABSTRACT (maximum 200 words) This thesis is in response to the annual requirement for departmental planning and scheduling of courses and instructors within all departments at NPS. This project thesis explains the process of analyzing, designing and implementing a web-enabled database capable of providing an effective and efficient tool for departmental planners. Using standard systems analysis procedures, this thesis provides a definition of the current business process, establishes an entity-relationship diagram for the desired process, constructs an operable database using MySQL, and provides a web-enabled interface for the population of data elements, creation of annual plans and reports for the extraction of decision making information.			
14. SUBJECT TERMS Information, Systems, IS, database, management, system, DBMS, DBM, entity-relationship, ER diagram,, E-R diagram, relational, model, development, develop, design, process, re-engineering, reengineering, MySQL, structured query language, SQL, myPHPadmin.			15. NUMBER OF PAGES 107
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UU

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

**ANALYSIS AND DEVELOPMENT OF A WEB-ENABLED PLANNING AND
SCHEDULING DATABASE APPLICATION**

Gary L. Reed
Lieutenant, United States Navy
B.A., Hampton University, 2004

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN INFORMATION TECHNOLOGY MANAGEMENT

from the

**NAVAL POSTGRADUATE SCHOOL
September 2013**

Author: Gary L. Reed

Approved by: Glenn L. Cook
Thesis Advisor

William J. Robinette
Second Reader

Dan Boger, PhD.
Chair, Department of Information Sciences

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

This thesis is in response to the annual requirement for departmental planning and scheduling of courses and instructors within all departments at NPS. This project thesis explains the process of analyzing, designing and implementing a web-enabled database capable of providing an effective and efficient tool for departmental planners. Using standard systems analysis procedures, this thesis provides a definition of the current business process, establishes an entity-relationship diagram for the desired process, constructs an operable database using MySQL, and provides a web-enabled interface for the population of data elements, creation of annual plans and reports for the extraction of decision making information.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I.	INTRODUCTION	1
A.	BACKGROUND	1
	1. Original Database	1
	2. Statement of the Problem	1
	3. Assumptions	2
	4. Methodology	3
	5. Organization of Thesis	3
II.	DECISION SUPPORT AND DATABASE MANAGEMENT SYSTEMS (DSS/DBMS)	5
A.	DECISION SUPPORT SYSTEMS (DSS)	5
	1. History	5
	2. Definition	7
	3. Classifications	8
B.	DATABASE MANAGEMENT SYSTEMS (DBMS)	10
	1. DBMS Classification (Data Models)	10
	2. DBMS Languages	12
	3. Types of DBMS	13
	a. Flat File Model	14
	b. Relational Model	16
	c. Hierarchical and Network Models	17
	d. Object-oriented Model	19
C.	DATABASE ARCHITECTURE	21
	1. Architectural Importance	21
	2. Single-Tiered (Centralized)	21
	3. Two-Tiered	22
	4. Three-tiered and N-tiered	23
D.	BEST SELECTION FOR PROPOSED DATABASE	24
	1. Choosing a Data Model	24
	2. Choosing an Architecture	25
	3. Designing the Database	25
III.	BUSINESS PROCESS ANALYSIS	27
A.	RE-ENGINEERING	27
	1. Considerations in Re-engineering a Process ...	27
	2. Purpose for Re-engineering This Process	27
B.	CURRENT BUSINESS PROCESS	28
	1. Identifying Critical Elements	28
	2. Course Information (First Set of Elements) ...	28
	3. Faculty Information (Second Element)	31
	4. Yearly Offering Information (Third Element) ..	31
	5. Analysis	32
C.	DESIGNING PROPOSED DATABASE	34
	1. Database Organization	34

2.	Operational Design	39
D.	INSTALLATION OF DBMS TOOLS	42
1.	Installing MySQL	42
2.	Installing Macintosh, Apache, MySQL, and PHP (MAMP) Package	44
E.	PROPOSED DBMS DESIRED CAPABILITIES	46
1.	Accessibility via Internet	46
2.	Importing Capabilities	46
3.	Cost	47
4.	Performance	47
5.	Schedule	48
F.	PROPOSED DBMS ENVIRONMENT	49
1.	Operating System (OS)	49
2.	Computer Resources	49
3.	Restrictions	50
IV.	PROPOSED OPERATIONAL FUNCTIONS AND CAPABILITIES	53
A.	GETTING STARTED	53
1.	Data Description Language (DDL)	53
2.	Opening MAMP	54
3.	Generating Database	57
a.	Generating Tables	60
b.	Generating Fields	61
B.	MANAGING THE DATABASE	63
1.	Data Manipulation Language (DML)	63
2.	Adding Records	64
3.	Editing Records	65
4.	Deleting Records	66
5.	Importing Records	67
C.	VIEWS & REPORTS	71
1.	Creating Views	71
2.	Reports	74
V.	CONCLUSIONS AND RECOMMENDATIONS	77
A.	CONCLUSION	77
1.	Solution	77
B.	RECOMMENDATIONS	78
1.	Future Application	78
2.	Additional Research	78
	APPENDIX	81
	LIST OF REFERENCES	87
	INITIAL DISTRIBUTION LIST	91

LIST OF FIGURES

Figure 1. Taxonomy of Knowledge (From Zeleny, 1987)	6
Figure 2. Simple Flat File Example (From Wikipedia, 2011) ...	15
Figure 3. Larger Flat File Model Example (From Dhesi, 2011) .	16
Figure 4. Example Hierarchical Model (From Zak, 2008)	18
Figure 5. Network Data Model Example (From MapsofIndia.com, 2009).....	19
Figure 6. Object Data Model Example (From Nordbotten & Crosby, 1999).....	21
Figure 7. General Steps Toward Database Design	26
Figure 8. Current Data Element Matrix	28
Figure 9. Sample Scheduling Spreadsheet	30
Figure 10. Sample Section Count	33
Figure 11. Proposed Data Element Matrix	35
Figure 12. Propose Database ER Diagram	38
Figure 13. Proposed Database Relational Schema	41
Figure 14. Tabs on MySQL.com Home Page (2011)	43
Figure 15. Platform Options on MySQL.com Download Page (2011).....	44
Figure 16. MAMP & MAMP Pro Home Page (2011)	45
Figure 17. MAMP Startup Interface with Servers Stopped	55
Figure 18. MAMP Startup Interface with Servers Running	55
Figure 19. MAMP Preference Options	55
Figure 20. MAMP Menu Tabs	55
Figure 21. MAMP Main Menu	56
Figure 22. MAMP Start Page within Internet Browser	56
Figure 23. Sample Layout of PMA Application within MAMP	57
Figure 24. Database Design Steps (Completed Steps Lined Out)	58
Figure 25. Proposed Database Generation DDL Script (Example)	59
Figure 26. PMA Home Button Option	60
Figure 27. PMA Quick Access to Create New Database	60
Figure 28. PMA Quick Access to Create Database Table	61
Figure 29. PMA Field Creation Options	61
Figure 30. Proposed DBMS <i>Offerings</i> Table DDL	63
Figure 31. PMA Insert Tab for Record Adding	64
Figure 32. PMA Comment (Insert Tab)	65
Figure 33. PMA Insert Tab (Completed Fields)	65
Figure 34. INSERT <i>Faculty</i> Table DDL Script Example	65
Figure 35. PMA Browse Tab Options	66
Figure 36. Faculty Table UPDATE DDL Script Example	66
Figure 37. DELETE DDL Script Example	67
Figure 38. LOAD DATA DDL Syntax Template (From MySQL, 2011) .	67
Figure 39. Sample of PMA Import Tab Wizard	68
Figure 40. Original Database Sample (Courses.xls)	68

Figure 41. Original Database Sample (Courses.csv)	69
Figure 42. Proposed Database Sample Post-Import Courses.csv ..	69
Figure 43. PMA Import DDL Script Using INSERT Function	70
Figure 44. PMA Import DDL Script Using LOAD DATA Function ...	71
Figure 45. PMA Create View Option	72
Figure 46. PMA Create View Screen Option	73
Figure 47. CREATE VIEW Syntax Template	74
Figure 48. PMA Query Tab	74
Figure 49. PMA View Placement (Example)	74

LIST OF ACRONYMS AND ABBREVIATIONS

=!	Not equivalent
1:1	One-to-one
1:M	One-to-many
AI	Artificial Intelligence
CBIS	Computer-based Information System
CODASYL	Conference on Data System Languages Model
CPU	Computer Processing Unit
CSV	Comma Separated Values
DBA	Database Administrator
DBMS	Database Management System
DDL	Data Definition Language
DML	Data Manipulation Data
DoD	Department of Defense
DSS	Decision Support System
EDP	Electronic Data Processing
EIS	Executive Information System
ER	Entity-Relationship
ESS	Expert Support Systems
Excel	Microsoft Excel
HSM	Human Systems Management
HTML	Hypertext Markup Language
ISS	Intelligence Support System
KBDSS	Knowledge-based Decision Support System

KMS	Knowledge Management System
LAN	Local Area Network
M:1	Many-to-one
M:M	Many-to-many (each sets equal cardinality)
M:N	Many-to-many
Mac	Macintosh
MAMP	Macintosh, Apache, MySQL, and PHP
MSS	Management Support System
MBP	MacBook Pro
MIS	Management Information System
OLAP	Online Analytical Processing
ODBMS	Object Database Management System
ODMG	Object Data Model Group
OO	Object-oriented
OODBMS	Object-oriented Database Management System
OS	Operating System
OSS	Open Source Software
PC	Personal Computer
PDF	Portable Document Format
PMA	phpMyAdmin
RDBMS	Relational Database Management System
SQL	Structured Query Language
TPS	Transaction Procession System

ACKNOWLEDGMENTS

I would like to acknowledge the numerous individuals who have supported me; many, indirectly, with moral support or simply allowing for an appropriately conducive environment; others, more directly, who provided information, resources, or opportunities for me to complete my objective.

Specifically, I would like to acknowledge Conner, Harris, Walton, and Jones from a personal standpoint.

To my database professor, Matthew Kolb, thanks for taking an intricate subject and making it much simpler. You provided an excellent foundation and understanding on which to build. To my thesis advisor, Glenn Cook, thank you for your infinite patience and foresighted instruction. I think you already know how this might have turned out without your assistance.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. BACKGROUND

1. Original Database

Departments at the Naval Postgraduate School (NPS) are required to plan and orchestrate the scheduling of courses which will be offered within their respective set of curriculums. One scheduling tool in use is the basic Excel spreadsheet. This is handy and easily accessible tool though, however powerful, is not meant to serve as much more than a simple database. The limitations associated with this form of scheduling tool can make managing the database very tedious, monotonous and error prone.

2. Statement of the Problem

Scheduling of courses occurs on a continuous basis. The planning portion includes future projections in budgeting, tracking, administration, and the like. The current process is error prone and redundant. Streamlining the process seems like the obvious solution. Though streamlining is obvious in the end state, it is not quite so obvious in the best path to follow to achieve that state.

One possibility is exploring the benefits of a whole new tool that could make this process easier to facilitate. Another may be just simply making revisions to the system already in place. Regardless of which is chosen, there will eventually be the question of cost-benefit analysis. This is basically determining whether any of the solutions are

worth the time, currency, and effort (i.e., cost) required to yield the foreseeable benefits that solution offers.

If achievable, the profits may be further reaching than just the immediate problem. Such changes would not only make the aforementioned improvements to the current system, but would also lend the characteristics necessary to support future growth of the database, if not also supplying broader implementation opportunities for similar systems.

However, if a cost-effective solution is not discovered, the current system will remain in place along with all of the ailments that lead to such endeavors for an alternative.

3. Assumptions

The first assumption is that additional scheduling tools, outside of the current solution, need to be explored. Considering that this current scheduling tool has been in use for years and is still causing operational angst despite numerous iterations in re-engineering suggests a different solution may be in order.

The second assumption in conducting this project thesis is that it must be done quickly. The longer it takes to develop the less practical it becomes, especially when considering possible use beyond the original database or original user. This assumption primarily pertains to the application of the re-engineering process to the original database and the transition time required for migration of content over to the re-engineered database. There is some expectation of additional time associated with research,

design, and fine-tuning. This time is, however, outside of the scope of this assumption.

The third assumption is that the solution will be cheap or free. This is not an explicit requirement, however, sets a developmental bearing at finding the best possible solution for the least amount of investment.

The fourth assumption is that the system will be fully functional of the Macintosh Operating System. This is the platform of the original database and the platform of the testing system being used to evaluate the solution.

The last assumption is that the solution will be remotely accessible via the internet. This adds a layer of convenience that shadows the desire for a more efficient and streamlined process.

4. Methodology

The process that will be followed in finding a solution for the current database management system will coincide with the principles of business process re-engineering: (1) Identify processes, (2) Review, update and analyze as-is, (3) Design to-be, (4) Test and implement to-be, and (5) Repeat (Harlan, 2009).

5. Organization of Thesis

The thesis will be organized by first reviewing the history of decision support systems and database management systems, the category into which this scheduling tool and future solution fall. Then there will be some discussion of what tools are available to meet the criteria expected of the system solution and, of those, which is best for the

job. After this will be an analysis of the current database and processes it utilizes followed by an analysis of the proposed solution's system. Last will be an in-depth look at the capabilities of the proposed solution as well as step-by-step guidance on how to utilize its features to manage the proposed system.

II. DECISION SUPPORT AND DATABASE MANAGEMENT SYSTEMS (DSS/DBMS)

A. DECISION SUPPORT SYSTEMS (DSS)

1. History

The concept of a decision support system (DSS) is one born of a combination of research in organizational decision making and technical research to develop interactive computing systems (Keen & Scott-Morton, 1978). It falls under the umbrella of computer-based information systems (CBIS), which also consists of automated systems, processing systems, management support systems, and so on (Eom, 2001).

One version of the four-stage model describing the evolution of knowledge starts with *data* and gradually makes its way to *wisdom* (or expertise). The simplest of these stages being *data*, results from nothing more than observation. Next is *information*, which seeks to collect, organize, store, and make data retrievable. Third is *knowledge*, which seeks to make sense of the organized data (or information) and find patterns and relationships; this is the realm of DSS. Last is *wisdom*, which attempts to contrive explicability from the knowledge for the sake of making a judgment call (Eom, 2001). Figure 1 depicts a matrix of this four-stage concept and its characteristics according to Zeleny (1987).

		<i>Technology analogy</i>	<i>Management</i>	<i>Metaphor</i>
Data	EDP	Elements: H ₂ O, yeast bacteria, starch molecules	Muddling through	KNOW - NOTHING
Information	MIS	Ingredients: Flour, sugar, spices, fixed recipe for bread only (OR/MS) type	Efficiency (Measurement + search)	KNOW - HOW
Knowledge	DSS, ESS, AI	Choose among different recipes for bread	Effectiveness (decision making)	KNOW - WHAT
Wisdom	HSM, MSS	Why bread and not croissant	Explicability (judgment)	KNOW - WHY

Figure 1. Taxonomy of Knowledge (From Zeleny, 1987)

The evolution of DSS takes us through the stages of knowledge evolution and mirrors the progression of CBIS. Transaction processing systems (TPS) and electronic data processing (EDP) were some of the first CBIS. These systems processed data in a very simple and straightforward manner. Examples of some EDPs would be mailing list, banking or point-of-sales transactions, or even purchasing a plane ticket or older payroll systems. Management information systems (MIS) go beyond the basic data tracking of data towards organizing the data in a meaningful way. This process converts raw data into information. These types of systems allow the management of information via functions such as generating reports. Annual performance reports, weekly sales reports, and inventory status reports are just a few of the reports more commonly seen on a regular basis. DSS, expert support systems (ESS), and artificial intelligence (AI) look to glean knowledge from the information collected. They present the information to the user based on some established set of principles programmed

into the system in order to support a shared knowledge within the human-computer interaction by which a more informed decision can be made. The final decision still lies with the user and simply draws upon the knowledge and experience the decision-maker brings along prior to engaging the system. Human systems management (HSM) and management support systems (MSS) round things up in still evolving attempts to actually have the system make decisions without the human decision-maker component (Eom, 2001).

2. Definition

Defining DSS is not quite as easy as the pairing done in tracing its evolution. Common definitions are very broad and cover a wide variety of technologies. All in all, the definition of a DSS is rather ill-defined (Shim, Warkentin, Courtney, Power, Sharda, & Carlsson, 2002). However, there seems to be a consensus that it involves a flexible computer-based solution to assist decision makers with unstructured and semi-structured problems (Reich & Kapeliuk, 2005). An important note is the recognition that it includes two sub-systems: the computer and the human decision maker (Eom, 2001).

Another thing that could be agreed upon about DSSs is that it is a tough beast on which to get a handle. Reich and Kapeliuk (2005) described DSSs as being some of the "most complex [information technology] products" in that they are developed for the express purpose of being interjected within core business processes simply to alter them. And since there is such an "abundance of technologies, tools and developmental methods" available, a

DSS developer's job does not actually become simpler but instead becomes more difficult in having to wade through these ever increasing options for the best tool to resolve his respective problem. The burden of such a task, knowing that there is no set approach, no tried and true solution, is therefore only compounded by additional confusion when coupled with the aforementioned inherently complex nature of DSSs.

3. Classifications

A classic DSS is equipped with data, information or knowledge; data management functionality to access that data; and a user interface that provides for querying, reporting, and data presentation via that data management tool. This simple understanding leaves classifying a DSS open to being determined by the type of data or problem on which the system is being utilized, and since their inception they have evolved quite a bit. This means classifications vary wildly, and from more specific to more general simultaneously.

More specialized examples, for instance, are the executive information systems (EIS) made to specifically assist senior executives to manage their organizations with timely, accurate, and filtered information (Elam & Leidner, 1993); knowledge management systems (KMS) to support the theoretical or practical assets obtained from the organization's data (Alavi & Leidner, 2001); and geographical information systems (GIS) designed as spatial decision support systems (SDSS) (Densham, 1991). More generalized examples in knowledge-based decision support systems (KBDSS), which are used to assist in a broad range

of functions from amplifying natural tools of the decision maker (intelligence support systems-ISS), or replacing human expertise for machine expertise yet still allowing the human decision-maker a choice in how to continue (ESS).

Furthermore, DSSs have made use of other technologies like data warehousing, online analytical processing (OLAP), data mining, and the web to broaden core functionality. And, yet, after three decades, this is still not the only taxonomy for classifying DSSs (Shim et al., 2002).

Hättenschwiler (1999) notes taxonomies at the user-level, conceptual-level, and technical-level. The user-level classifies DDSs by passive, active, and cooperative users. Passive provides assistance but gives no explicit suggestions. Active provides a suggestion. Cooperative allows the user the option.

The conceptual-level categorizes by communication-, data-, document-, knowledge-, and model-driven DSS as per *Decision Support Systems: Concepts and Resources for Managers* (2002) by Powers. Communication-driven focuses processes on sharing between multiple users much like Microsoft Groove or SharePoint, DropBox, or Google Docs. Data-driven is centered on accessing and manipulating data trending over time. Document-driven is oriented in providing a variety of formats in which to retrieve and manage the data. Knowledge-driven is focused on some established set of rules or guidelines which act as problem-solving machine expertise (Hättenschwiler, 1999).

Hättenschwiler also references Powers (1997), which breaks up DDSs between enterprise-wide and desktop DSS, illustrating the technical-level categorization.

Enterprise-wide is for large, linked companies with multiple managers. Desktop is for the single user operating on a small, individual system.

B. DATABASE MANAGEMENT SYSTEMS (DBMS)

1. DBMS Classification (Data Models)

In any database (DB) there exists a level of data abstraction. Data abstraction is the suppression, or hiding, of details allowing for better focus on the essential portions of the database. This feature makes the organization of the database easier for the user to understand (Elmasri & Navathe, 2007).

The level of abstraction can usually be specified to accommodate the level of sophistication preferred by the user. For instance, the average end user may just care for viewing the most basic level of abstraction, while a developer, designer or DB administrator (DBA) might desire to include far more detail. Data models provide this capability. Data models are a collection of representations describing the structure of the database. The structure generally refers to the type of data included, the relationship of the data, and the rules and constraints governing how the data is to behave within the database (Elmasri & Navathe, 2007).

Types of data models range from either of two data model extremes: high-level (conceptual) or low-level (physical) data models. According to Ambler (2002-2011) those that fall in-between are classified as logical, or a type of mid-level model. The mid-level model can also be called the representational (or implementation) data

models, which is the most frequently used type commercially (Elmasri & Navathe, 2007).

Conceptual models are those that more closely portray how the average person perceives data. This is for the typical end user. Classifying conceptual models is determined by entities, attributes, and relationships of the data. The entity-relationship (ER) diagram is a very popular example of a high-level conceptual model. Physical data models more closely portray how data is actually stored on the computer or storage mechanism. This is for the computer specialists. Classification for physical models is determined by how the data is stored; whether it be based on the record format, record order, or even the access path. A record refers to a set or grouped instance of data. Access path refers to the structure used to search for a particular record. An index is a type of access path that leads directly to an indexed term or keyword. It operates in much the same way as the indices within a book allow a reader to quickly find a topic (Elmasri & Navathe, 2007).

In a cinematic analogy comparing the conceptual and physical model differences, in *The Matrix*, one data model would be like being inside the matrix and the other like being outside of the matrix; allowing viewing of the underpinnings and how things really work. Right in between these two extremes exist the representational (implementation) model, which is a happy medium. End users can still understand it while the computer folks can still get a feel for the behind the scenes of it.

Data models are at the cores of every DBMS, and essentially act as a blueprint for developing the DBMS. DBMS are classified based on the data model used to develop them. As of 1979, there were more than 40 data models in existence, though most lacked a stable definition or complete fulfillment of the perceived role of data models at the time. A common point of confusion was the misunderstanding that data models were simply a collection of data structure types. Such notions failed to realize that operators and integrity rules were just as essential to the proper understanding of a structures behavior, and thus for complete definitions that allow clear distinctions between each (Codd, 1980).

The first data model to fulfill its role as a proper data model, primarily to manage formatted data, was the relational data model. The relational model falls under the representational side along with hierarchical, network, and object data models. The relational data model is very widely used amongst the representational models. Hierarchical and network are considered to be legacy models. And though object data models are also within the representational side of the house, they lie on the cusp of conceptual and are part of the newer family of higher-level representational models called the object data model group (ODMG) (Elmasri & Navathe, 2007).

2. DBMS Languages

After designing a database, the next thing to do is specify the conceptual and internal schemas. The DBMS languages are the tools that allow for such specifications.

Amongst these tools are the data definition language (DDL), storage definition language (SDL), view definition language (VDL), and data manipulation language (DML). These all makeup what is known as structured query language, or SQL (Elmasri & Navathe, 2007).

The internal schema refers to the physical parameters and specification related to data storage. This is implemented using SDL, which was part of SQL in early versions but removed to keep SQL at the conceptual and external levels focused on the data vice the storage performance or physical storage structures.

The conceptual schema refers to how data is stored within the DBMS with regards to the DBMS' data model. DDL is used to define this schema.

DDL can also be used to define the external schema, which is what specifies what views an end user has available to them. In a true three-schema architecture, or conceptual-external schema only architecture, VDL would be used to define the external schema (Elmasri & Navathe, 2007).

3. Types of DBMS

The DSS component this document will take a closer look at, depending on which definition of DSS consulted, is the DBMS. Even within the varying definitions of a DSS, the DBMS is one element that appears rather frequently. Typically, a DBMS performs data access, definition, structure, security, and recovery. The functions a DBMS must provide are data persistence, secondary storage management (e.g., database versus repository, data

concurrency, data recovery, and data definition and manipulation). Data persistence is a trait denoting that after the execution of a program the data should still exist. Secondary storage management speaks to the DBMS' ability to make changes to the non-primary memory, which generally does not offer direct execution, fetch, load, and store functionality. Data recovery is somewhat of a backup feature referring to the ability to salvage data after damage, corruption, or some similar event rendering it inaccessible by normal means. Lastly, there is data definition and manipulation; functions of the DDL and DML, which are used to instantiate, edit, and delete the data retained within the database.

The most common logical (or representational) data models on which a DBMS is based are flat file, relational, hierarchical, network, and object-oriented. However, since a flat file does not provide all of the services of a DBMS, it is not included amongst the most common types of DBMS. The missing functionality is mainly data naming, redundancy and concurrency control. It also requires the user to interact directly with the physical layout of the file (Minoli, 2008).

a. *Flat File Model*

Flat file databases are those that store data in a single file, or table (Trustees of Indiana University, 2006). They are usually plain text files containing one record per line and some distinctive character, known as a delimiter, separating the fields. They are by far the easiest of the databases to setup, however are not DBMS in

the truest sense; failing to provide concurrency control and measure again redundancy to name a few (Minoli, 2008).

One of the simplest examples of a flat file database would be construction of a database on a sheet of paper. Figure 2 provides a very basic example of this concept as pertaining to a team division roster (Wikipedia, 2011). With manually developed rows and columns, it would essentially be the same implementation as a flat file. It has the benefit of having a quick initial setup and requiring very little space or memory, in the case of a text file. However, the drawbacks include lost efficiency as the database size increases, greater risk of inconsistency and errors, and inconvenience; as user must know the exact location of the data desired in order to access it (Minoli, 2008). Figure 3 shows an example of a bit more involved flat file model for a sales order database (Dhesi, 2011).

id	name	team
1	Amy	Blues
2	Bob	Reds
3	Chuck	Blues
4	Dick	Blues
5	Ethel	Reds
6	Fred	Blues
7	Gilly	Blues
8	Hank	Reds

Figure 2. Simple Flat File Example (From Wikipedia, 2011)

Customer File						
Customer ID	Company Name	Contact Name	Address	City	Country	Phone
ALFKI	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	Germany	030-007431
AROUT	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	UK	(171) 555-7788

Employee File			
Employee ID	Last Name	First Name	Title
3	Leverling	Janet	Sales Representative
4	Peacock	Margaret	Sales Representative
6	Suyama	Michael	Sales Representative

Product File					
Product ID	Product Name	Category	Quantity Per Unit	Unit Price	Units In Stock
28	Rossie Sauerkraut	Produce	25 825 g cans	\$45.60	26
39	Chartreuse verte	Beverages	750 cc per bottle	\$18.00	69
41	Jack's New England Clam Chowder	Seafood	12 12 oz cans	\$9.65	85
46	Spegesild	Seafood	4 450 g glasses	\$12.00	95
52	Filo Mix	Grains/Cereals	16 2 kg boxes	\$7.00	38
63	Vegie-spread	Condiments	15 625 g jars	\$43.90	24

Order File							
Order ID	Customer ID	Employee ID	Order Date	Required Date	Shipped Date	Ship Via	Freight
10643	ALFKI	6	25-Aug-1997	22-Sep-1997	02-Sep-1997	Speedy Express	\$29.46
10692	ALFKI	4	30-Oct-1997	03-Oct-1997	31-Oct-1997	United Package	\$61.02
10793	AROUT	3	24-Dec-1997	21-Jan-1998	08-Jan-1998	Federal Shipping	\$4.52

Order Detail File				
Order ID	Product ID	Unit Price	Quantity	Discount
10643	28	\$46.50	15	25%
10643	39	\$18.00	21	25%
10643	46	\$12.00	2	25%
10692	63	\$43.90	20	0%
10793	41	\$9.65	14	0%
10793	52	\$7.00	8	0%

Figure 3. Larger Flat File Model Example (From Dhesi, 2011)

b. Relational Model

Edgar Codd (1980) claims that the relational model was, in fact, developed in 1969. Though, other sources have otherwise reported that relational models were developed in 1973 by, none other than, Edgar Codd (Minoli, 2008). This is likely the year that more formal definitions of data models were developed thus attributing to another possible misconception that hierarchical and network models preceded relational models. Actually, it is stated that while said systems were developed before 1970, respective data models were not defined until 1973 after having been abstracted from observations of those systems (Codd, 1980).

Like flat files, relational DBMS (RDBMS) utilize tables of data. Unlike flat files however, relational models make use of more than one table, which are related and thus connected through the underpinnings of mathematical theory of relations (Minoli, 2008).

Relational models are likely the most popular of the currently defined data models and are extensively used in the commercial sector. Still, the relational model is evolving. A more collaborative effort has been pursued recently. Though not listed amongst the common DBMS mentioned earlier, there is now an object-relational model in which are incorporated some concepts developed with object databases (Elmasri & Navathe, 2007).

c. Hierarchical and Network Models

Hierarchical and network models are very similar. The main differences pertain to the relationships maintained between each record stored. The hierarchical model has a tree-like structure consisting of one-to-many (1:M) relationships, also known as a set-type. This means that each parent record may have many children, or subordinate, records but each child record may have no more than one parent record (Minoli, 2008). Figure 4 gives an example of one such model for some language-related organization (Zak, 2008).

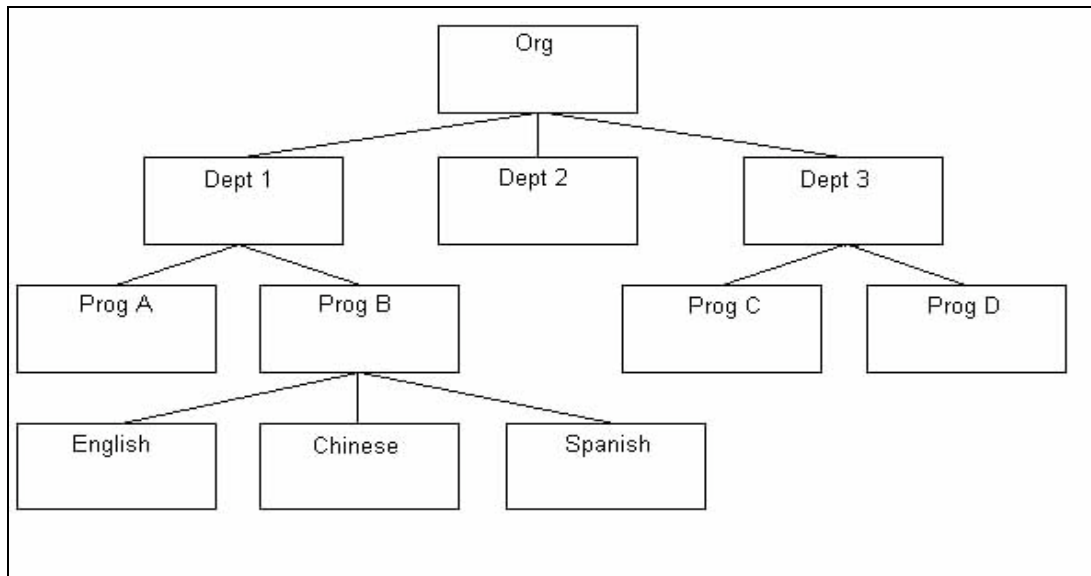


Figure 4. Example Hierarchical Model (From Zak, 2008)

The network model is also known as the conference on data system languages (CODASYL) model. It became popular around the same time as the hierarchical model. As with the hierarchical model, the set type is a basic component within the network model, thus they have a similar structural arrangement. The exception is that the child records in the network model are not limited to having only one parent record as depicted in Figure 5 (MapsofIndia.com, 2009). This type of multi-parent, multi-child relationship is called a many-to-many (M:N) relationship. This model is based on mathematical set theory (Minoli, 2008).

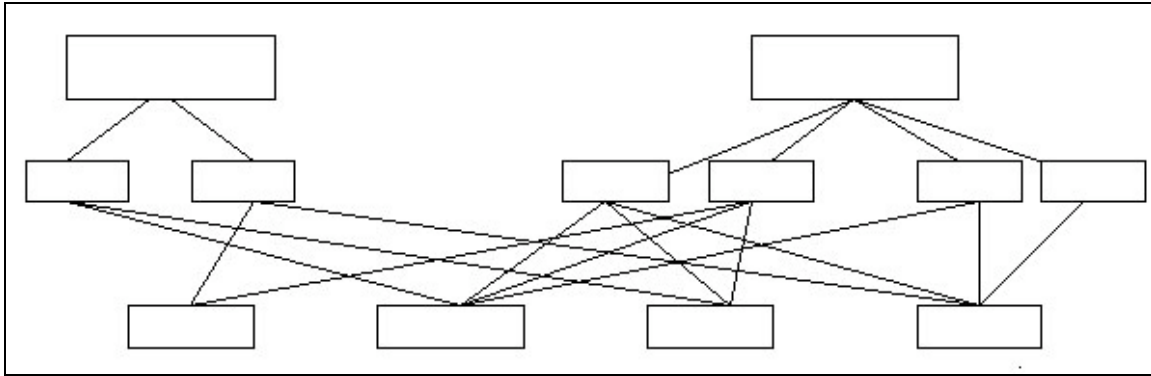


Figure 5. Network Data Model Example (From MapsofIndia.com, 2009)

Both, the hierarchical and network models, are historically important models recognized as legacy data models. These models still have an active following that includes banks and hospitals (Elmasri & Navathe, 2007). Though popular at one time, it is conjectured that these models were displaced by the relational model because their low-level navigational benefit was eventually surpassed by relational model productivity and flexibility combined with hardware speed advances over time.

d. Object-oriented Model

Object data models focus on managing objects vice just simple data (e.g., strings, integers, booleans). These objects are essentially constructed of attributes and methods. Attributes are the properties of the object. They define and, basically, describe the object and may be simple data or complex objects (e.g., objects containing more objects). Methods are operations, or functions, of the object. They are executable code describing the object's behavior (Kim, 1990). A third critical component is classes. Objects with the same structure and behavior

belong to the same class (Elmasri & Navathe, 2007). Classes are model helpers, object templates that a database can use to recreate or validate an object. They are setup in hierarchies called acyclic graphs. Each object can only belong to one class ("Object Oriented Databases," 2010). Figure 6 is an academic example of an OO model (Nordbotten & Crosby, 1999).

Object-oriented (OO) or, simply, object DBMS (OODBMS / ODBMS), though utilized in some commercial sectors, have had a difficult time catching on and gaining widespread use (Elmasri & Navathe, 2007). As Kim (1990) alluded to, object models are possibly one of the most confusing of the data models. Seemingly in direct competition with relational models, especially for the commercial market, the degree of confusion surrounding ODBMS very likely did not help.

ODBMS boast excellent features like data encapsulation, inheritance, overriding capability, and computational completeness through general-purpose language use (Minoli, 2008). Direct advantages over RDBMS even include reduced coding time, faster performance (i.e., execution time), better concurrency control, easier navigation, and easy internal integration of multimedia. Still RDBMS remain market leaders likely because greater efficiency with simple data relationships, greater upscaling capability at higher volumes, a greater numbers of tools already in existence (Whatever Happened to Object-Oriented Databases?, 2011), more stable standards, and adaptability (i.e., ability to add-on object model features

through software extensions, hence object-relational DBMS) (Object Oriented Databases, 2010).

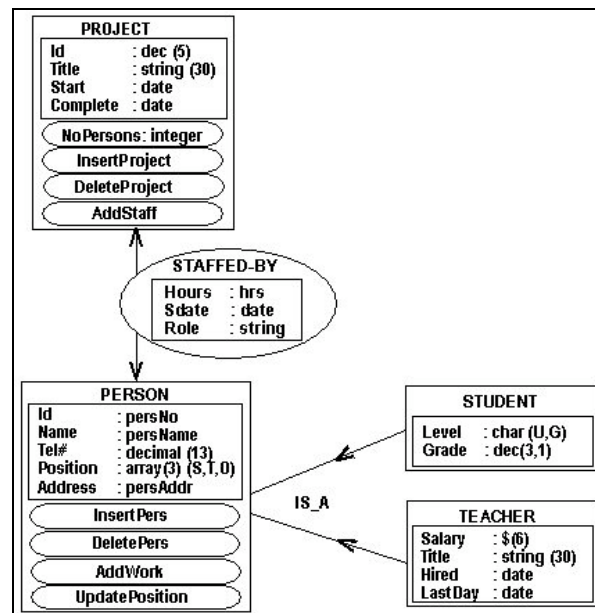


Figure 6. Object Data Model Example (From Nordbotten & Crosby, 1999)

C. DATABASE ARCHITECTURE

1. Architectural Importance

DBMS architecture has everything to do with performance. For this reason, this concept has gone hand-in-hand with computing power. In order to get the best performance out of the DBMS it simply meant having the best possible computer to manage, process, and distribute the data for display (Elmasri & Navathe, 2007).

2. Single-Tiered (Centralized)

Centralized DBMS architecture was the first method utilized to implement DBMS functionality. With no other connectivity outside of the computer mainframe and

terminal, users were at the mercy of the scenario aforementioned regarding computing processing power. This type of architecture is considered single-tiered because everything (i.e., user interface, DBMS operations, and application operation) took place on a single computer (Elmasri & Navathe, 2007).

3. Two-Tiered

Two-tiered architecture is known as the client/server architecture. When machines became smaller and more capable it yielded personal computers (PCs). Increased connectivity amongst those computers via networks followed shortly thereafter. The idea of file servers grew to use as specialized servers that eventually accommodated the specific needs of the DBMS. Early versions of this architecture still had all DBMS functionality provided by the server side except the user interface capabilities (Elmasri & Navathe, 2007).

Over time, ingenuity suggested moving more functionality to the client side. After interface capabilities, next came application programs (Elmasri & Navathe, 2007). Before this advent the setup primarily supported thin clients, or user side computers that simply depended on the server to provide all of the application processing needed to access the DBMS (Lai & Nieh, 2006). As the incorporation of two-tier DBMS on the commercial scene was increasing, movement from having thin-clients was also increasing. Sharing the burden data processing allowed organizations to provide service to more users in a more superior fashion. In short, it was good for business.

4. Three-tiered and N-tiered

The three-tier architecture took the two-tiered, client/server, architecture and added a third tier called the middle tier. This tier is sometimes known as the application server or web server. Adding this tier, in a way, relieved both the host and client from having to bear the load of application processing (Elmasri & Navathe, 2007). In the case of an application server, this solution more or less just separated the application programs from the database server, which still improved individual server performance (Kambalyal, n.d.).

The intermediate role played by the middle tier improved more than just performance, it also improved security. Moving, once again, back towards the thin client setup, end users only had the direct access to the interface capabilities that made requests of the DBMS via the application or web server. This protocol allows the middle tier to also conduct a credential check before forward the user's request to the database server (Elmasri & Navathe, 2007).

N-tier architectures do much the same as three-tier architectures but add one or two tiers making the final tier count four or five. These additional tiers are typically used to further divide the business logic layer. This further relieves any processing burdens of a server or possibly provides the additional tiers as specialized servers within the same system. These additional servers are then free to run the appropriate operating systems (OS) or use the appropriate processors to meet the needs of that particular middleware layer (Elmasri & Navathe, 2007).

D. BEST SELECTION FOR PROPOSED DATABASE

1. Choosing a Data Model

The primary end user of this database first and foremost desired more than simply a database, which is what was being provided by the current process. The most desired functionality essentially made managing the data in the database more automated. This meant concurrency control, for instance not having to track, search for, and individually change every instance of a professor's or course's name simply to update the database. This meant developing views such that already inputted data could seamlessly be used to generate reports that included the necessary formatting and calculations. This meant being able to access and safeguard this data at the same standard or higher than was already being implemented, but including the potential to expand said features to remote management.

These features were accompanied with two other important restrictions: time and technical considerations. The end user needed to be able to implement the new system within a set amount of time in order to replace the current system. Next, the end user needed to avoid the technical side of managing the database, thus allowing this tool to increase productivity without having to exert a significantly larger amount of time and effort.

Understanding all of these factors places plans directly in-line with the relational model. Avoiding the technical side of managing the data suggests using a conceptual approach in development. The data models conducive to a conceptual approach are the relational and object-oriented models. Knowing that only simple data is

planned to be stored in the rather small database favors the relational approach over the object-oriented. Lastly, recognizing that the DBMS needed to be developed quickly sealed the choice of a relational data model in creating the proposed database.

2. Choosing an Architecture

In this area of designing the proposed DBMS, the centralized architecture was actually working just fine. All DBMS functionality was intended to be operated and maintained on a single computer. Deciding to use a two-tier architecture with an internet browser interface was more of a consideration for extensibility. For, using a two-tier architecture does not take away from the proposed DBMS' capabilities. This just means that there will be a logical separation between the database server and the database application on the host computer; in much the same way as would be seen with most client/server setups.

Furthermore, having the interface provided through an internet browser adds even more extensibility, for instance to a three-tier architecture should it be decided to make a web server to link remote access to the host computer-database.

3. Designing the Database

After determining the data model and architecture comes the designing of the database. A top-down design method was chosen for this step in design. The top-down design method is also known as design by analysis (Elmasri & Navathe, 2007). Having all the data and information pertaining the current database (i.e., entities,

attributes, structure, and relations) this method seemed the more beneficial and obvious method to select. From this development of the new database schema can begin.

The steps following are to transition from the current process to the proposed. In its entirety this is process re-engineering and new process implementation. Progress can now be marked by milestones that generate a piece of the blueprint that will result in the proposed database sought. Figure 7 shows these milestones and their products.

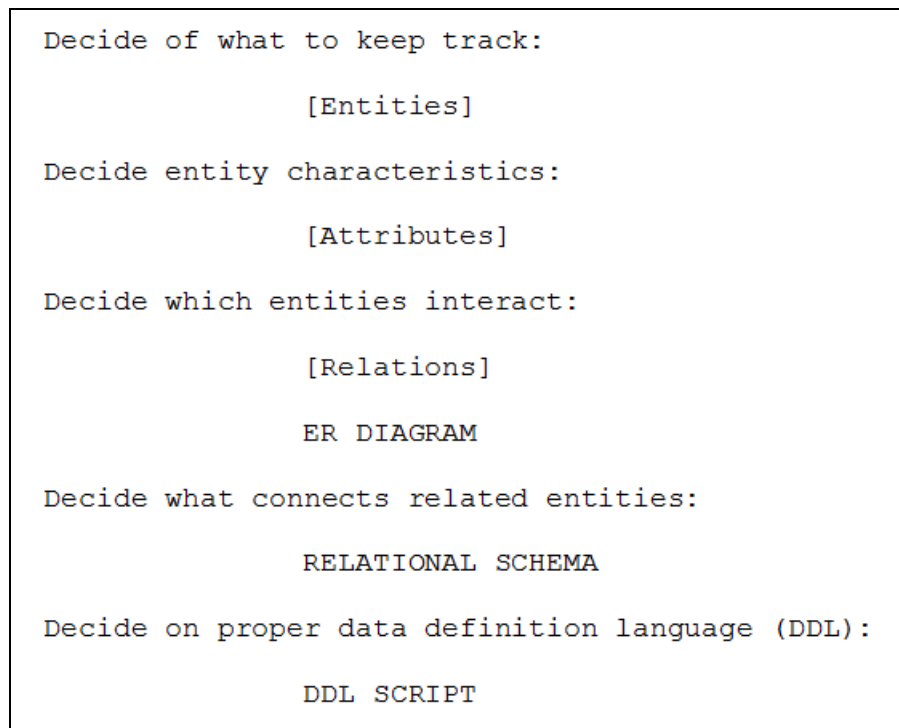


Figure 7. General Steps Toward Database Design

III. BUSINESS PROCESS ANALYSIS

A. RE-ENGINEERING

1. Considerations in Re-engineering a Process

When looking to re-engineer any process, first take a close look at the original process and determine what is occurring within that process. The questions asked before taking on the task of re-engineering that process are essential in deciding the best solution in redeveloping the process. Some of those questions are:

- "What are the basic elements of the current process?,"
- "How do those elements relate to each other?,"
- "How do those elements interact with each other?,"
- "Within the process dynamics, what elements should be mutable by users? Which by administrators?,"
- "How are each of the elements to be referenced again or put to use outside of the current data storage?"

2. Purpose for Re-engineering This Process

This project seeks to create a web-enabled scheduling database from the current scheduling tool, which primarily consists of a scheduling spreadsheet. The current process is highly manual, very time-consuming, and prone to error. These characteristics are the key drivers in reconstructing the scheduling tool. By correctly identifying the "moving parts" the components necessary to make the new database more automated, a bit less laborious, and more robust or resilient in the event of error.

So, before asking the key questions that guided the development of the new scheduling database, first looking at the elements and interactions between those elements within the current process is recommended.

B. CURRENT BUSINESS PROCESS

1. Identifying Critical Elements

The most general categories of the current system are Course Information, Faculty Information, and Yearly Offering Information. Each consists of their own set of elements (Figure 8).

Course Information	Faculty Information	Yearly Offering Info
Course Number	Last Name	Fiscal Year
Course Name	First Name	Quarter
Lecture Hours	Tenure Status	
Lab Hours		
Course Format		
Funding Source		
Course Curriculum		

Figure 8. Current Data Element Matrix

2. Course Information (First Set of Elements)

Course Information is composed of the *Course Number*, *Course Name*, *Lecture Hours*, *Laboratory Hours*, *Course Format*, *Funding Source*, and *Curriculum*. The *Course Number* is the primary way of identifying the course in question. This is a unique combination of the curriculum abbreviation and number code denoting the level and subject of the course. This is most often delineated with a two-digit curriculum abbreviation (e.g., IS, and a four-digit number code [e.g., 3202]). However, the current system is flexible enough to accept deviations in the naming convention.

An example of a course number would be IS3202, which is a 3000-level course offered within the Information Systems (IS) curriculum identifying the Web-Enabled Database Management & Development Course, the course name.

Though, both the *Course Number* and the *Course Name* are supposed to be unique in their designation, the current system is not setup to enforce this restriction. This is most likely due to limited size of the current database as well as the somewhat extensive amount of extra work required to arrange this type of enforcement protocol on a Microsoft Excel spreadsheet.

Lecture Hours and *Laboratory Hours* denote the number of hours devoted to subject lecturing or spent within the laboratory setting per week. This is tracked in order to determine how much time is required to administer the course as well as better indicate how much effort will be required of the professor.

Course Format is the manner in which the course is being administered. This refers to whether the course is conducted on campus (resident) or through distance learning (online or via remote site). This is another category mainly managed by expert knowledge of scheduling system. Within Figure 9, of the two categories available, only distance learning courses are obviously marked. To an expert user, however, it is known that the "courses" and "electives" are resident courses and, as the norm and majority, require less noticeable indications than those that are not the norm.

Course	Lecture	Lab	Name	FALL AY 2011	WINTER AY 2011	SPRING AY2011	SUMMER AY2011
AY2011				Demand (sections)	Demand (sections)	Demand (sections)	Demand (sections)
Courses							
AB2020	2	3	Beginners Course				
AB3200	3	2	Intermediate Course		ProfB		ProfB
AB3202	4	2	Another Intermed Class		ProfC		
AB4031	4	0	Upper Lvl Crs	ProfA		ProfA	
AB4220	4	2	Harder UL Class		ProfD		ProfD
AB4300	3	2	Remaining Course	ProfE		ProfE	
Electives							
AB3210	4	0	Elective One			ProfF	
AB4188	3	2	Elective Two				
DL Courses							
BC3000	4	0	My Kind of Class		ProfG		ProfH
BC4920	4	0	Home Schooling				
CD3200	3	2	Proxy Course				Profil
Tenure Track				0	2	1	1
Non-Tenure Track				1	1	1	3
Military Faculty				1	1	1	0
Total				2	4	3	4

The *Funding Source* refers to how the course is paid for. This can be either directly funded or reimbursed funds. In the current process, this is primarily determined by what type of faculty is administering the course. This characteristic is only noticeable in observing the automatic adjustment of funding statements due to a pre-entered equation. This basic, built-in system functionality is used to make this operation more automated. Though, not needed in the current system, the system does maintain the ability to expand the aforementioned functionality to the introduction of additional funding sources.

3. Faculty Information (Second Element)

Faculty Information is composed of the faculty member's *Last Name*, *First Name*, and *Tenure Status*. The *Last Name* and *First Name* are rather self-explanatory and are the primary means of uniquely identifying each faculty member. The current process' system relies on expert knowledge in the actual course scheduling outlay. Classes are listed with no more than faculty member's last names, while full first and last name info is listed on a separate spreadsheet. The two instances of faculty listings are not connected (i.e., selecting one does not pull from the listing of the other) nor is the uniqueness requirement enforced. Such functionality can be inputted with a less than intuitive and user-friendly manner, and maintaining its fidelity is quite a bit more involved with regards to setup and upkeep.

Tenure Status is a denotation of whether the faculty member is on a tenured track or a non-tenured track. This applies to civilian faculty only.

4. Yearly Offering Information (Third Element)

Yearly Offering Information is made up of the *Fiscal Year*, and *Quarter*. *Fiscal Year* is the four-digit number denoting the year the course is scheduled to be offered. This is based-on the academic year starting in September of each year. To better illustrate, the courses offered from September 2010 to December 2010 would be categorized as the part of the first quarter of the 2011 fiscal year.

Quarter is either of the four seasons over which a course is scheduled within three-month intervals of the

year (e.g., winter, summer). Following the typical understanding of the fact that the year consists of four quarters, there are four options to choose from being that courses are offered year-round.

5. Analysis

The current process uses very little automated functionality. The spreadsheets used in the original process, like depicted in Figure 9, are developed using Microsoft Excel, which offer some automated functionality most suited for aiding in running tallies and calculations. One such automation is found in the calculation of total faculty utilized per quarter and annually. The numbers are first broken up by either of the two tenure track status or military. These are not listed and assigned a value that the spreadsheet would account for in counting how many of each are used, but rather counted by hand, entered manually and from there the counted numbers are totaled via a cell formula option. This is likely due to the disproportionate amount of time and effort required to input and track individual instances in comparison to the perceived return on investment, which is no more than a simple staff count. This same reasoning is likely used with regards to the funding spreadsheet (Figure 10), which is similarly inputted and tallied.

Section Count	
Tenure Track	Department Funded
ProfD	2
ProfE	2
ProfH	1
ProfI	1
Total TT	6
MILFAC	Department Funded
ProfA	2
ProfG	1
Total MILFAC	3
Non-Tenure Track	Department Funded
ProfB	2
ProfC	1
ProfF	1
Total NTT	4
Total Sections	13
Total Paid Sections	10

Figure 10. Sample Section Count

In order to archive courses offered previously along with information about those courses (i.e., faculty teaching, number of sections offered, course number, lecture and lab hours) multiple versions of the same spreadsheet are saved under a different file name denoting what time period it covers. Using this type of sub-process has advantages and disadvantages. It requires a lot of attention in maintaining an organized filing system. It also exposes the risk of having incomplete archives due to improper filing (e.g., accidental overwriting, misplaced documents) or errant naming conventions. On a more positive side, having each record maintained separately guards

against total loss of archives in the event of file corruption or contamination.

Each of the aforementioned data elements has been deemed critical in maintaining the functionality of the scheduling system. So, keeping them and their interactions in mind, the re-engineering of the original process may begin with the express goal of adding the convenience of automation, remote access, and increased user friendliness.

C. DESIGNING PROPOSED DATABASE

1. Database Organization

After identifying the essential elements of the current business process and identifying the additional elements desired in the new system, all the tools necessary to start designing the proposed database are available.

The first step is developing an entity-relationship (ER) Diagram. This not only lays out the elements in their most basic form but, provides a great visual blueprint to how each element relates to the next within the larger scheme of the database.

In the ER diagram created for the proposed database, the three main categories of the original database (Course Information, Faculty Information, and Yearly Offering Information) were maintained. The two additional categories (i.e., Curriculums and Tenure) were created to accommodate the desired elements that were not previously being provided or tracked in the current database. Figure 11 shows a new element matrix with asterisks next to the sub-elements post re-engineering.

These new elements are *Course Coordinator*, *Course Price*, and *Operational Status* in relation to Course Information. Then there is *Academic Rank*, *Qualifications*, *Preferred Quarters*, and *Operational Status* in connection with Faculty Information. Yearly Offerings Information now includes *Section Number* and *Location*. *Cost* was placed under Tenure Info and a *Requirement Status* was added in association with Curriculum Information.

Course Information	Faculty Information	Yearly Offering Info	Tenure Info*	Curriculum Info*
Course Number	Last Name	Fiscal Year	Cost*	Requirement Status*
Course Name	First Name	Quarter	Type	Associated Curriculum
Lecture Hours	Tenure Status	Section Number*		
Lab Hours	Academic Rank*	Location*		
Course Format	Course Qualifications*			
Funding Source	Preferred Quarters*			
Course Coordinator*	Operational Status*			
Course Price*				
Operational Status*				
Course Curriculum				

Figure 11. Proposed Data Element Matrix

Course Coordinator is the sole faculty member responsible for coordinating the material covered in a particular course. Each course must be assigned a coordinator and only one faculty member may be selected. Despite the number of sections or faculty members facilitating the course, the course coordinator must be recognized as being a separate and equally vital role.

Next is *Course Price*. This denotes the cost of the course based on whether it is taught by a tenured, non-tenured. Military faculty members fall under the classification of non-tenured for the sake of determining the *Course Price*.

Course operational status references whether the class is being offered during a specified time period; usually

active scheduling. This element is intended to allow filtering in options in selecting or displaying courses, not to necessarily for long-term tracking. An example of a course status would active or inactive.

Academic Rank refers to the position held by the faculty member: full professor, associate professor, assistant professor, research professor, senior lecturer, lecturer, or military.

Course Qualifications refers to the courses a faculty member is qualified to teach vice simply the courses a member may be scheduled to teach.

Preferred Quarters indicates when the faculty members desire to teach courses being offered.

Faculty Operational Status denotes whether a faculty member is active or inactive during the current scheduling period. This is not tracked long-term but used to assist in active scheduling of courses.

Section is the number indicating how many classes of a particular course need to be offered in order to accommodate the demand in student requests. For instance, if there are 50 students who need to take a class in a certain quarter, two sections of that course many need to be offered to keep the number of students per classes down to a reasonable amount.

Location refers to where each class is held. This data element is made up of two items of data, the two-digit building abbreviation and three- or four-digit classroom number to include a letter. Two examples would be IN263 or R0200C.

Tenure Cost indicates the monetary amount required to offer a course with regards to the type of faculty teaching the course.

Curriculum Requirement Status is an element to indicate whether a particular course is required for a specified curriculum or merely an elective.

Each of the preceding elements described fall under a particular category (e.g., Course Information). In the proposed database, the elements generally maintained their grouping under similar categories, or umbrella subjects. Each of these subjects is known as an entity. These entities mirror the data contained within them respectively. The individual data items, previously referred to as data elements, within each entity, are known as attributes. The break down and organization chosen for this proposed system is depicted in the diagram shown in Figure 12.

The entity *Courses*, abbreviated *CRS*, is the focal point for the other entities. *CRS* are arranged to have a one-to-many (1:M) relationship with *Course Offerings*, abbreviated *Offerings*, meaning each course offering will only consist of one course, but each course can be used in more than one offering. For instance, section one of IS3202 would be considered one offering and thus is made up of one course. However, IS3202 could be offered again, this time as section two or some subsequent section within the same fiscal year and quarter.

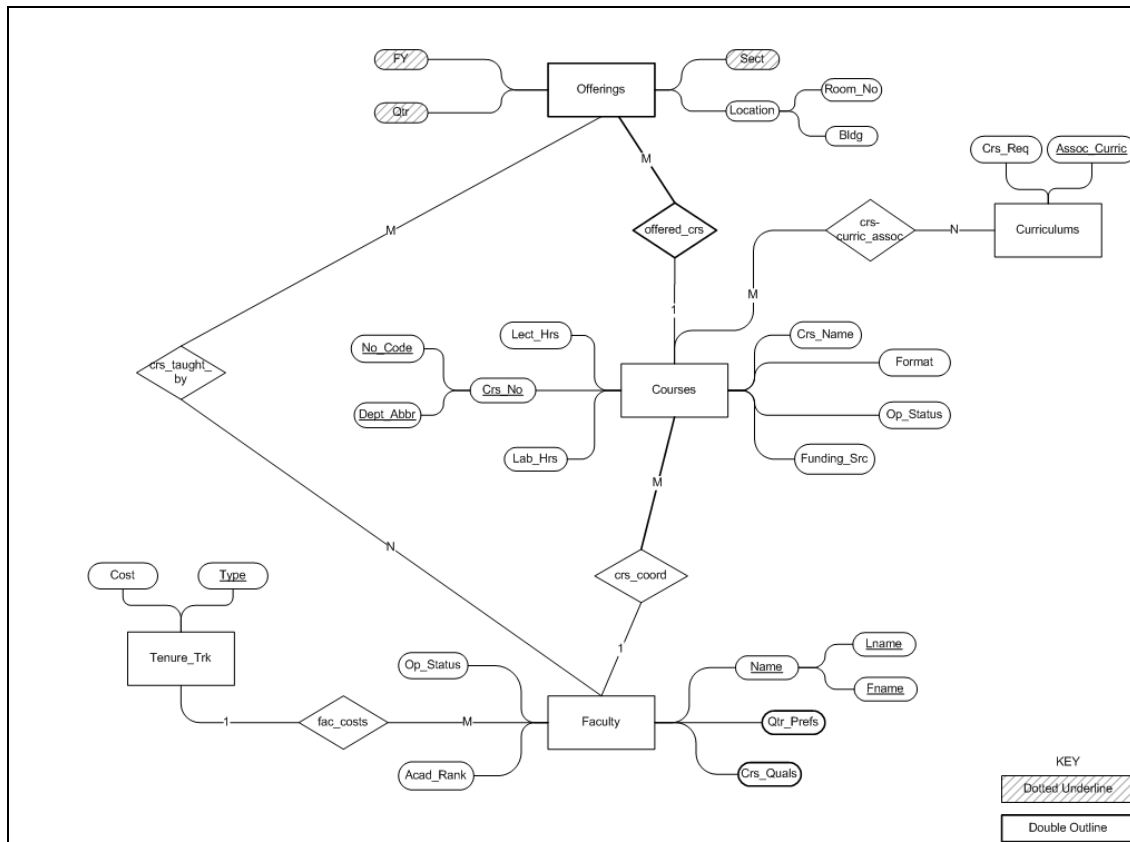


Figure 12. Propose Database ER Diagram

Another characteristic of this *CRS-Offerings* relationship regards its constraints. Notice in Figure 12 there exists bold lines around the dependent entity, *Offerings*, and the respective *CRS-Offerings* relationship, *offered_crs*. These, in fact, represent double outlines, as per the key, and moreover represent that it is mandatory for each offering within the database to have a course with which it is associated. It also asserts that this is not the case for courses, as they can exist within the database regardless of whether they are being offered.

Faculty also has a relationship with *CRS*. This relationship is two-fold and stems from the distinction

mentioned earlier between course professors and course coordinators. The *CRS-Faculty* relationship pertaining to course professors is a many-to-many (M:N) relationship, meaning each course can be taught by more than one faculty member and each faculty member can administer more than one course. The *CRS-Faculty* relationship pertaining to course coordinators is a many-to-one (M:1) relationship, meaning each course can only have one coordinator but each faculty member can act as a coordinator for more than one course.

Familiar bold markings, representing a double line, connecting *CRS* to the *crs_coord* relationship denote that this is a mandatory relationship for *Courses*. Each course must have a course coordinator, which is selected from among the faculty. However, this double line representation is not between *crs_coord* and *Faculty* because it is not mandatory that all faculty to course coordinators.

The *CRS-Curriculums* relationship is a M:N relationship. Each course can be associated with multiple curriculums or can exist within the database without an association. Likewise, curriculums can consist of multiple courses however do not require them in order to be instantiated within the database.

The *Faculty-Tenure_Trk* relationship is a M:1 relationship denoting that faculty may only have one tenure track but may a one of the available tenure tracks can be assigned to more than one professor.

2. Operational Design

The next step in designing the proposed database is determining how each entity is connected to the other and

how best to represent each relationship within the database. This can easily be considered the most conceptually difficult portion of designing a database, as there are no steadfast rubrics for deciding how the model should interact with itself—just general rules of thumbs that can usually narrow best practices down to a couple sound choices.

Figure 13 illustrates how the proposed database is designed to interact. Each row represents a table within the database. Each of the entities are listed and will have their own tables.

Relationships within a database are usually represented by migrating an attribute from the related entity's table and placing it within the partner entity's table. The relationships within the proposed database that received their own table, vice the aforementioned arrangement (e.g., *crs_taught_by*), are permitted so due to the type of relationship to which they refer. These kinds of objects are indicated by lines within Figure 13 beginning with lower-case titles. *crs_taught_by* is a M:N relationship which must have its own table. *crs_coord* is a M:1 relationship which warrants a separate table but can just as justly be represented with a data attribute from *Faculty* referenced within the *CRS* table. The type of data attribute being referred to is called a foreign key (Elmasri & Navathe, 2007). The former approach was chosen in Figure 12. The latter approach would be illustrated by subtracting the *crs_coord* row and rewriting the *CRS* row as **"CRS(Crs_No, Crs_Name,..., Funding_Src, FacName"** where *FacName* represents the *Name* attribute in the *Faculty* row.

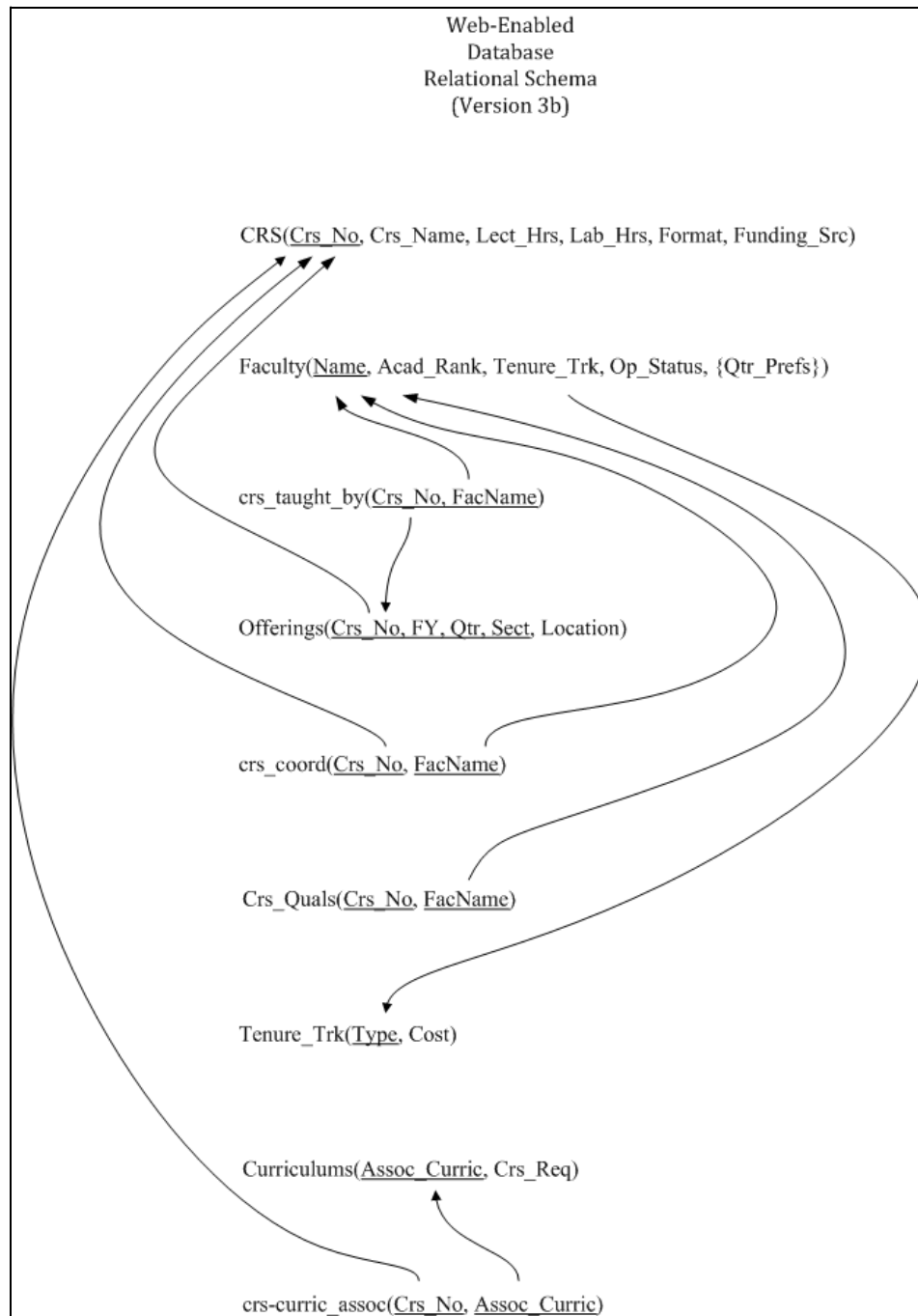


Figure 13. Proposed Database Relational Schema

Attributes are also generally included within the table of the associated entity. The attribute included in the proposed database's relational schema with its own

table was *Crs_Qual*. This is a multi-valued attribute much like *Qtr_Prefs*. Both can be represented as separated tables or as some finite number of attributes within their parent entity's table (Elmasri & Navathe, 2007). *Crs_Qual* was chosen as a separate table due to the unknown and potentially unbounded number of courses a professor can be qualified to teach. *Qtr_Prefs*, on the other hand, is limited by the numbers of quarters there are in a year and thus represent a perfectly reasonable number of distinct attributes including amongst the partner table's attributes.

The completed relational schema acts as a roadmap for directly developing the database. The names for each of the entities and attributes were thus chosen accordingly. Consideration was given to length, intelligibility of abbreviations, uniqueness, and avoidance of reserve words within the SQL programming language. The next step is generating the appropriate SQL script, or syntax, from the relational schema, which is best done after selecting and installing the SQL processing software of choice. For the proposed system the software chosen is MySQL.

D. INSTALLATION OF DBMS TOOLS

1. Installing MySQL

There are several key reasons MySQL was chosen as the database management solution for this particular web-enabled scheduling system: cost, performance, and schedule. Each of these will be discussed in more detail later in the chapter.

After having made the decision to use MySQL, the next step was installing the management tool. To acquire the software go to the MySQL homepage at <http://mysql.com> (Figure 14).

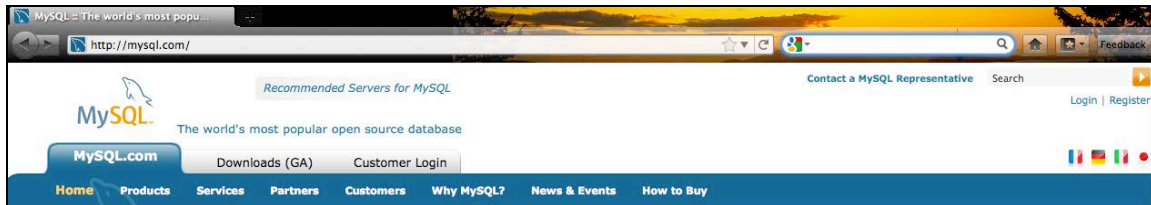


Figure 14. Tabs on MySQL.com Home Page (2011)

Click on the 'Downloads (GA)' tab at the top of the page. Then scroll down, find the 'DOWNLOAD' under the MySQL Community Server heading and select it.

Next scroll down and select the appropriate platform from the dropdown menu, followed by selecting the 'Download' button next to the correct machine description (Figure 15).

Version 5.1 was used to develop the proposed system. From here follow the step-by-step instructions for running and installing.

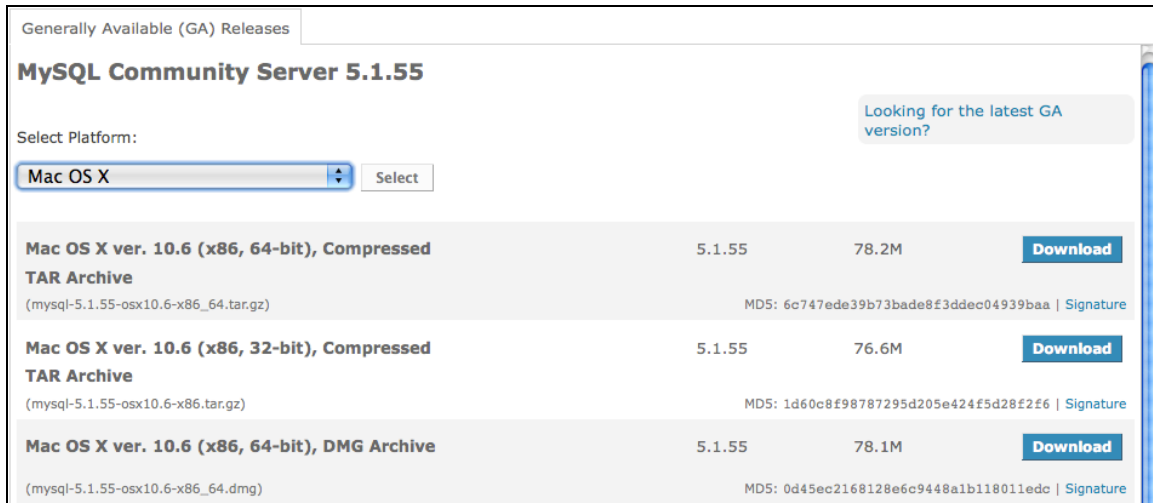


Figure 15. Platform Options on MySQL.com Download Page (2011)

Now, to install a graphical user interface (GUI) tool that will improve the ease of use and general user's understanding of the MySQL application. To acquire this program, return to the MySQL Homepage. Select the 'Downloads (GA)' tab again and scroll down to the MySQL Workbench (GUI Tool). As before, select 'DOWNLOAD' under this heading. Then choose the correct platform followed by the 'Download' button next to the appropriate machine description.

After installation, the basic tools necessary for running SQL scripts are available.

2. Installing Macintosh, Apache, MySQL, and PHP (MAMP) Package

MAMP is the application software package chosen to integrate all of the tools needed to attain a database server, online extension, and a user friendly GUI for database management.

MAMP; which stands for Macintosh, Apache, MySQL, and PHP, still utilizes MySQL as the SQL processor. It uses PHP as the SQL web extension language, which also interacts with the database for access, maintenance, and manipulation. And Apache is the web server software. The proposed DBMS was developed with MySQL 5.1, PHP 5.2, and Apache 2.0 versions running on version 1.9.4 of MAMP.

To acquire this application, visit the MAMP Homepage at <http://www.mamp.info> (Figure 16). Then scroll down to click on the 'Download now' button beneath the MAMP personal web server logo.

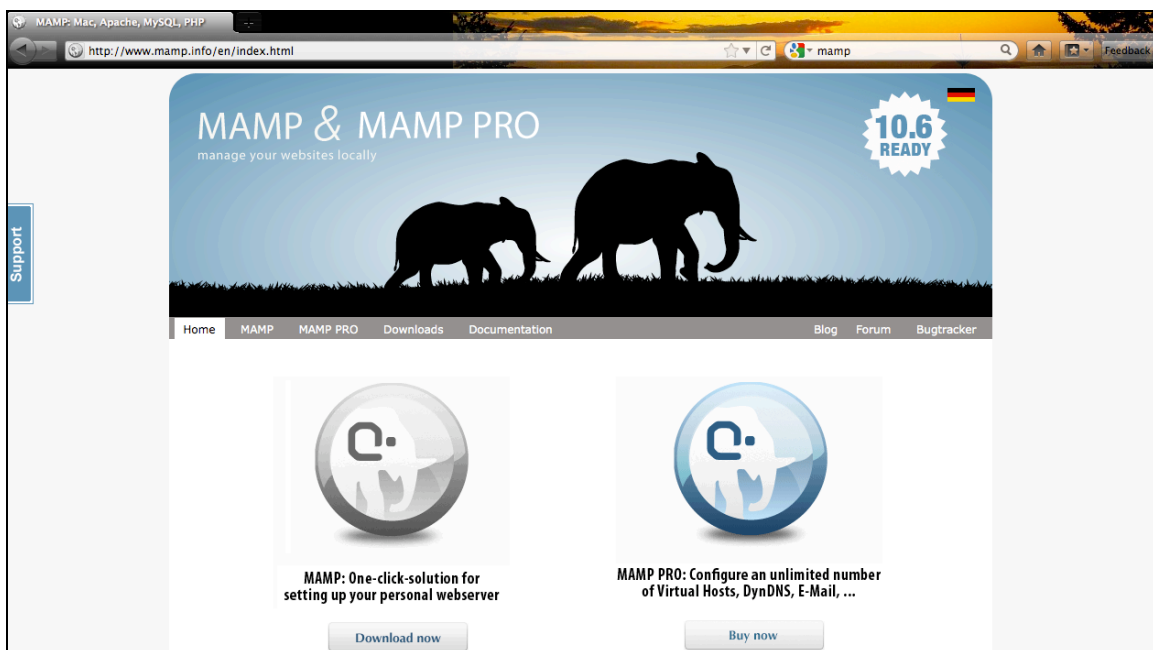


Figure 16. MAMP & MAMP Pro Home Page (2011)

From here simply follow step-by-step directions and first session and server startups can be initiated upon completion.

E. PROPOSED DBMS DESIRED CAPABILITIES

1. Accessibility via Internet

An important feature driving the development of the propose DBMS is the capability to access and manage the system via the web. This capability maintains a critical factor of convenience, which is available in the current business process; a feature that should not have to be sacrificed in order to supply the additional capabilities requested.

PHP is a general-purpose scripting language designed for web development and embedding in hypertext markup language (HTML) (The PHP Group, 2001). This is the language chosen to extend the MySQL-based DBMS, adding the aforementioned functionality.

2. Importing Capabilities

It can be a hassle to have to reenter all of the data from even the smallest database into another database. Importing ideally allows verbatim capturing of data in the new database with far more convenience and less likelihood of error resulting from reprocessing. The general language exists within SQL to allow such functionality, however the particulars may vary depending on the SQL processing program being used.

MySQL contains a LOAD DATA function that provides the importing capability. With the current system's data being stored within an Excel spreadsheet, an .xls file, MySQL's LOAD DATA function is available to import data saved as a .csv (comma separated value) file (Oracle, 2013).

When saving an Excel file as a CSV file, the values entered within the spreadsheet are essentially saved to a form of text file with all the cell data separated by commas (Import Excel Data, 2011). Lines or rows within Excel are separated within the CSV file by some other standard character. These characters are known as delimiters (MySQL, 2011).

Several SQL processors, including MySQL, can read and process documents like CSV files. With some user input, these processors can properly interpret the data and place it within the appropriate tables in the importing database.

3. Cost

One huge driving factor in this re-engineering project was cost; a criterion that has been well accommodated.

MySQL is not only one of a handful of database applications available on the Mac, but is also free. Yes, it is free software, as in freeware, being that the source code was developed under the GNU General Public License. However, MySQL has both definitions of 'free' covered; free in the developer's rights in distributing source code and free of financial requirements (Kennedy, 2010).

MAMP is another freeware application bundle that integrates additional open source software like MySQL. Apache and PHP both fall into this category of software.

4. Performance

For such a great price on the actual tools implemented to build and management the propose DBMS, it might be expected that there would be some tradeoff in quality or

functionality. This is, however, not the case with the tools chosen. MySQL and Apache alone have made significant marks on their respective industries.

MySQL has become increasingly more popular in the development of the web applications. In fact, it boasts of having exceeded 100 million downloaded copies since its first release up through version 5.1 (Kennedy, 2010). Such a positive outlook on the software can very likely be attributed to the low tradeoff of power and quality despite the price. Moreover, as the product further increases in popularity, there will be an increase in developers focused on making the most of such capable, open source software (OSS).

Apache is credited with playing a significant role in growing the web (Netcraft, 2011). And while remaining to be freeware, was the first web server software to be used by over 100 million web sites (Netcraft, 2009).

Neither is likely to have gained such widespread use if the necessary quality were not provided. Consider that the world seems to be riddled with free products, especially software, which still cannot find a niche in mainstream usage.

5. Schedule

Schedule is another big driver, especially in this particular DBMS project. The proposed system has a very practical use that could be applied on a much bigger scale. However, considering the rather limited size of the current system data, a product that takes a great deal of time to

create does not seem "cost effective" in terms of benefit versus effort or time required

The bulk of the time spent in bringing the proposed database to fruition was accrued in the conceptual planning and documentation of the system. The building of the database took significantly less time, especially when the documentation was well organized. This trend suggests that with some familiarization with the tools used and some expert assistance in areas of inexperience, there could be a drastic increase in the time efficiency ratio and thus even more increase in returns on investment (ROIs).

F. PROPOSED DBMS ENVIRONMENT

1. Operating System (OS)

The proposed database and the current scheduling system are being run on an Apple iMac. This is running Mac OS X version 10.6, also known as Snow Leopard. It is essential that the proposed system be able to run and be effectively managed via this machine with the aforementioned OS.

Testing and development of the proposed system were done on an Apple MacBook Pro running Mac OS X version 10.5, also known as Leopard, with incremental verification done using the iMac on which the system is intended to run.

2. Computer Resources

Both systems used in the development and operation of the proposed DBMS, the MBP and the iMac respectively, have very similar specifications. Both are fairly new machines in the personal computer (PC) marketing industry. Both have

Intel processors, allowing such convenient functions as running a Windows OS on each as virtual machines (VM). The MPB is dual-core while the iMac is quad-core. This will not cause a problem since the computer used to develop the database actually has less processing power than the intended operation-computer. Additionally, tests have shown that MySQL still performed well on limited-resource computer processing units (CPUs) running OSs like Fedora and Ubuntu (Ahmed, Uddin, Azad, & Haseeb, 2010).

The hard drive (HD) capacity was not deemed negligible, as the proposed solution required less than 200 MB to install both the free and proprietary versions. Slightly more HD space is required for the sake of computer processing and data storage, however considering the limited size of the current data, each machine maintained more than enough space for running and managing the proposed DBMS.

Both computers are also utilizing versions of Mozilla Firefox web browsers and have access to the Macintosh (Mac) native browser, Safari. The desired web browser for managing the system is Firefox, the second most widely used web browser according to February 2011 W3Counter Global Stats and possibly first on Apple computers since Internet Explorer's development for Mac was discontinued in 2003 (Dairymple, 2003).

3. Restrictions

The most pressing restrictions in developing all the desired system capabilities are (1) availability of preferred DBMS interfacing software and (2) availability of the data to authorized users from remote sites. These

restrictions are primarily due to the location of the data being accessed by the proposed DBMS. The iMac on which the system will be operating is connected to a Department of Defense (DoD) owned and managed local area network (LAN). This characteristic does not affect the functionality of the DBMS' data management operation while working locally, at the original computer for which it was designed. And, although MySQL databases can be managed offline, meaning without being connected to a network or internet source, the desired functionality includes access and management via internet and more specifically includes compatibility with Firefox web browser.

It appears that the variety of software applications on DoD networks can be quite limited. With respect to the validity of such an assertion, it would not be unheard of that programs like Firefox might not be available on a government-owned Apple Computer being that Firefox Browser is not native to Macs. If not for this reason alone, it is nice to know that such restrictions in operation due to variations in web browsers are mitigated by the universal nature of the DBMS chosen. It not only functions with Firefox 3.6, the version being run by the iMac, but it also works with Firefox 4.0 beta and Safari 5.0, which are being run on the MBP and were used for testing and development.

The next restriction considers the network firewall(s) and virtual private network (VPN) protocols. Direct access from a remote computer to the host computer, the iMac, via the DoD network would not be available.

An alternative method for gaining access would be via the network's VPN. This detracts from some of the systems

intended convenience in that remote access to the system would be consequent on the proxy computer's acquisition of the DoD network's VPN software and permissions granted through a login account in addition to whatever security measures are added to the DBMS by its administrator. However, on a positive note, it supplements this drawback with increased security merely through association with the DoD network. This has in itself the added convenience of security that is already in place and is maintained by resources other than those directly added in the development of this system.

Yet another alternative to local access and management would be hosting the database on a web server existent outside of the DoD LAN, which is accessible to the host computer and authorized remote computers. There are numerous free or commercial web hosting service providers available from which to choose. This can also be done from a remote computer, for instance a home computer.

IV. PROPOSED OPERATIONAL FUNCTIONS AND CAPABILITIES

A. GETTING STARTED

1. Data Description Language (DDL)

DDL is the DBMS scripting language that will be use to generate and define the proposed database. The primary uses of the DDL are to access the database for the sake of adding data, modifying data, or deleting data. The most common SQL commands for accomplishing these tasks are: CREATE TABLE, CREATE INDEX, ALTER TABLE, RENAME TABLE, DROP TABLE, and DROP INDEX (Elmasri & Navathe, 2007).

Other noteworthy features in the realm of DDL are constraints. For the moment, focus will be directed towards ON UPDATE / ON DELETE, which defines the behavior of data in one table that is linked to data in another table with what is called a foreign key. This feature allows for four options if that particular data instance is modified or deleted. Those options are: CASCADE, SET NULL, SET DEFAULT, or NO ACTION (or RESTRICT). CASCADE changes the dependent data instance to reflect the change as it appears in the parent table. SET NULL makes the value of the data null, or undefined, in the dependent table. SET DEFAULT changes the value of the dependent data whatever value is set as the default in the table's definition of that particular attribute. NO ACTION and RESTRICT are essentially the same (Oracle, 2012). These commands leave the data in the dependent table the same as before the change was made in the parent table. In standard SQL, using RESTRICT may result in an error message as well as reject the change in the parent table. If using NO ACTION, the change would

still not occur in the dependent table but may not result in an error message or stoppage of change in parent table; making this, for all intents and purposes, like unlinking the data, except in their namesake. In MySQL, since constraints are checked immediately, the update or delete operation would be rejected for the parent table regardless of using NO ACTION or RESTRICT.

2. Opening MAMP

After having installed the chosen DBMS, run it and configure it to perform the tasks for which it was selected. To do this, search for the DBMS, MAMP, within the applications folder. From here, double click on MAMP to execute the run code. Following these actions should result in display of the startup interface for MAMP. This interface is depicted in Figure 17 with the servers stopped. Figure 18 depicts MAMP with the servers on. As a default, configuration may be set for MAMP to 'Start Servers when starting MAMP' and 'Open start page at startup.' These options (shown in Figure 19) can be accessed inside of the preferences menu located under the MAMP tab (Figure 20 and Figure 21).

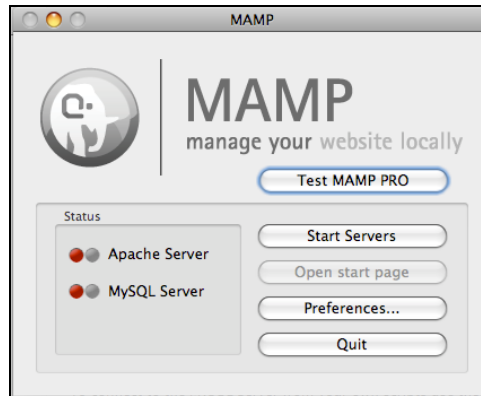


Figure 17. MAMP Startup Interface with Servers Stopped

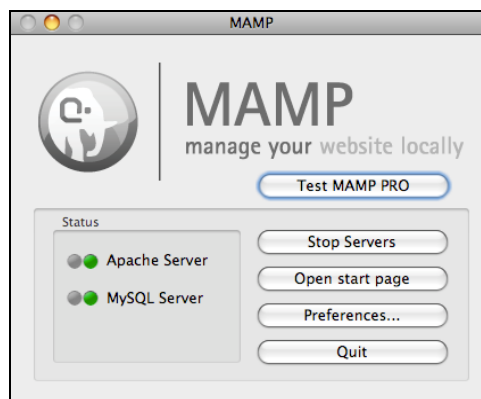


Figure 18. MAMP Startup Interface with Servers Running

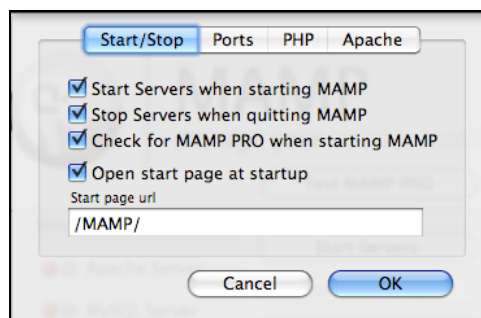


Figure 19. MAMP Preference Options

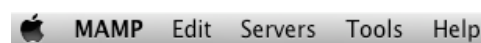


Figure 20. MAMP Menu Tabs

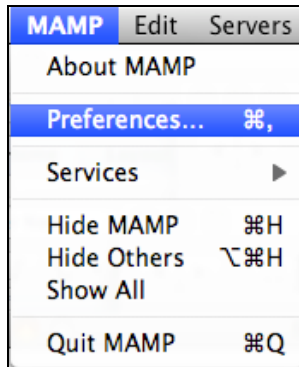


Figure 21. MAMP Main Menu

Once the DBMS servers are running, the start page will either open automatically or will require selection of the 'Open start page' button on the startup interface. This action will open the specified internet browser on the MAMP start page (Figure 22). MAMP is designed to operate within the host's default browser. This allows for seamless configuration in using MAMP remotely via the Internet.

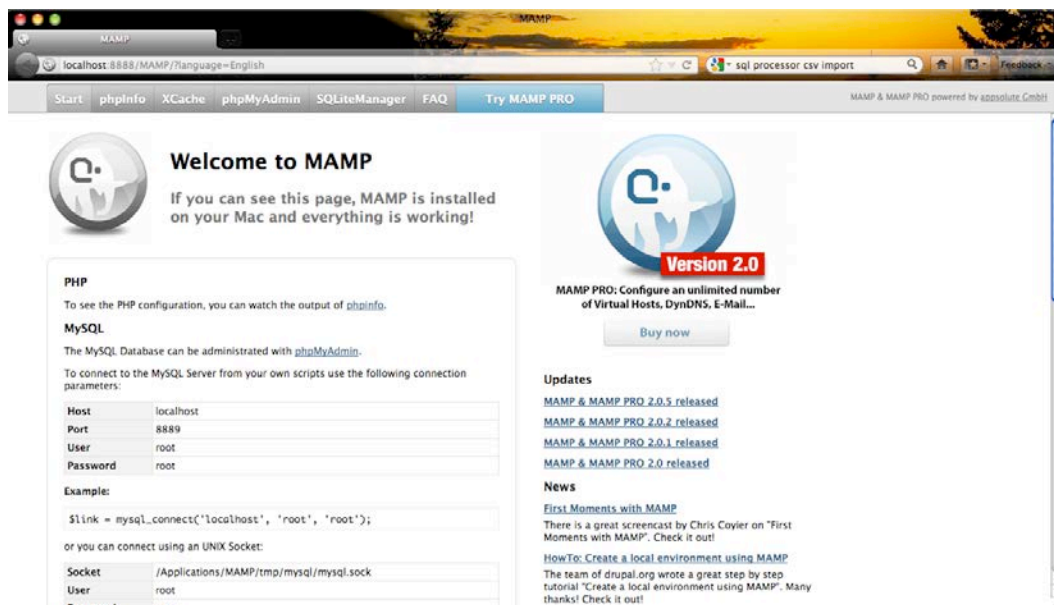


Figure 22. MAMP Start Page within Internet Browser

From here, the primary way to interact with the database is through phpMyAdmin (PMA). Click on this tab in order reach developer options. This will be where the database is setup & configured. Figure 23 shows an average layout of a database within PMA.

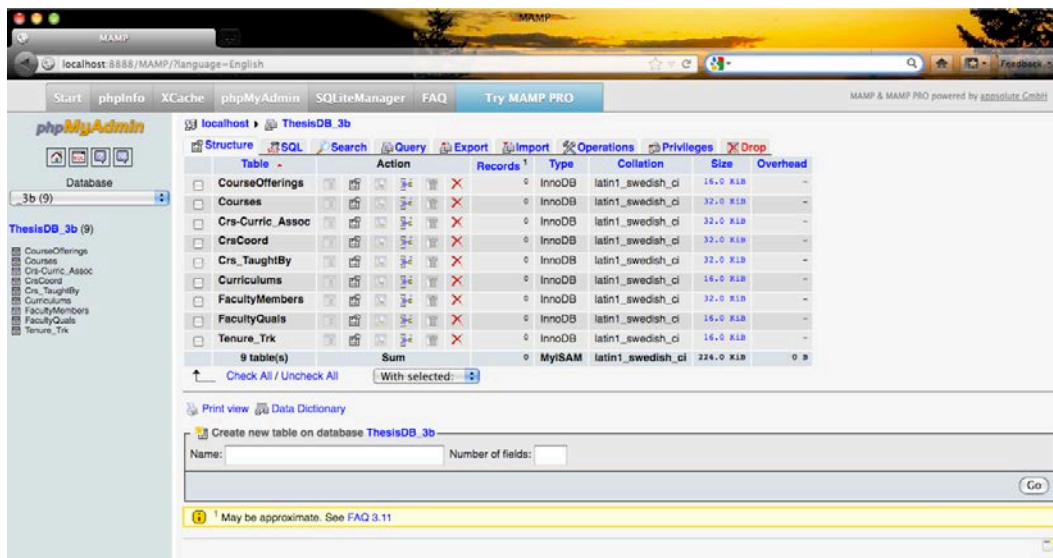


Figure 23. Sample Layout of PMA Application within MAMP

3. Generating Database

Up to this point the road map for how to develop a database has been followed through to the last step. This road map and the steps included within it are revisited in Figure 24, indicating what has been completed up to this point.

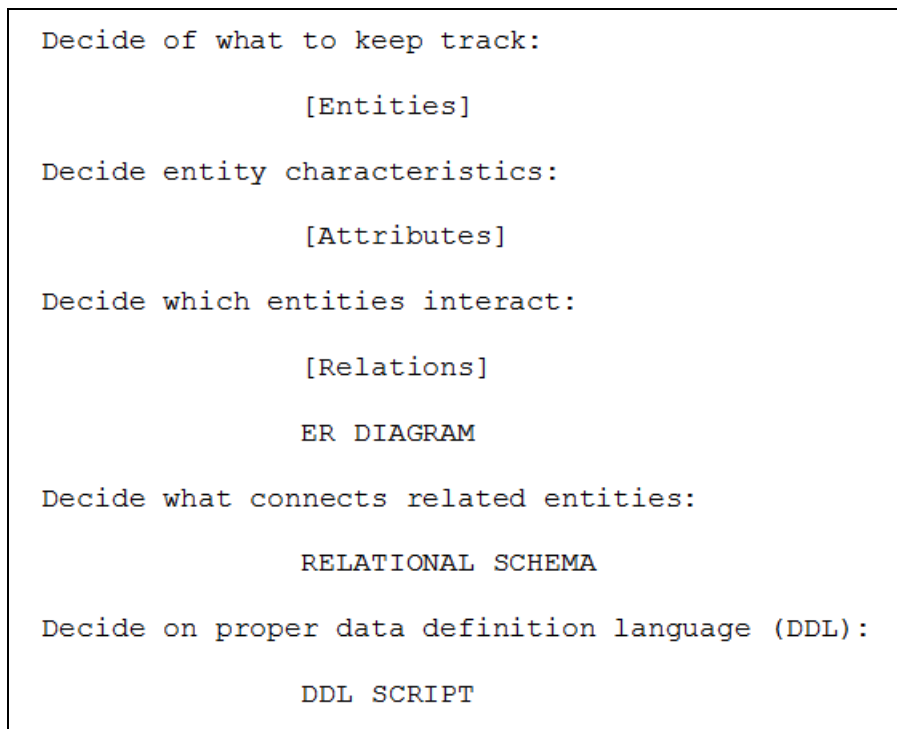


Figure 24. Database Design Steps (Completed Steps Lined Out)

The next step is using the relational model to determine the script necessary to generate the proposed database. This script is called data definition language. It will be used to define the data as it is to exist within the database. Figure 25 is an example of one such script for the proposed database. However, SQL has some flexibility in the formatting of this script, which allows for a bit more variation than some traditional programming languages. Regardless of this feature, SQL does still have firmly defined syntax to which must adhered.

```

DROP SCHEMA IF EXISTS WebSkedDB;
CREATE SCHEMA WebSkedDB;
USE WebSkedDB;

CREATE TABLE Course (
    Crs_No          VARCHAR(8)          PRIMARY KEY,
    Crs_Name        VARCHAR(40)         NOT NULL,
    Lect_Hrs        INTEGER,
    Lab_Hrs         INTEGER,
    Format          VARCHAR(10),
    Funding_Src     VARCHAR(12) )
ENGINE = InnoDB;

CREATE TABLE FacultyMembers (
    Lname          VARCHAR(20)          NOT NULL,
    Fname          VARCHAR(20)          NOT NULL,
    Acad_Rank       VARCHAR(20),
    Tenure_Trk      VARCHAR(3),
    Op_Status       VARCHAR(7)          NOT NULL,
    1stQtr_Pref     BOOLEAN,
    2ndQtr_Pref     BOOLEAN,
    3rdQtr_Pref     BOOLEAN,
    4thQtr_Pref     BOOLEAN,

    CONSTRAINT Faculty_pk PRIMARY KEY (Lname, Fname)
) ENGINE = InnoDB;

CREATE TABLE CourseQuals(
    FacLname        VARCHAR(20),
    FacFname        VARCHAR(20),
    Crs_No          VARCHAR(8),

    CONSTRAINT Crs_Quals_pk PRIMARY KEY (FacLname, FacFname, Crs_No)
) ENGINE = InnoDB;

CREATE TABLE TaughtBy(
    Crs_No          VARCHAR(8),
    FacLname        VARCHAR(20),
    FacFname        VARCHAR(20),

    CONSTRAINT taught_by_pk PRIMARY KEY (Crs_No, FacLname, FacFname)
) ENGINE = InnoDB;

CREATE TABLE CourseOfferings(
    Crs_No          VARCHAR(8),
    FY              INTEGER,
    Qtr             CHAR(3),
    Sect            INT,
    Location         VARCHAR(7),

    CONSTRAINT Offerings_pk PRIMARY KEY (Crs_No, FY, Qtr, Sect)
) ENGINE = InnoDB;

CREATE TABLE CrsCoord(
    Crs_No          VARCHAR(8),
    FacLname        VARCHAR(20),
    FacFname        VARCHAR(20),

    CONSTRAINT CRS_Coord_pk PRIMARY KEY (Crs_No, FacLname, FacFname)
) ENGINE = InnoDB;

```

Figure 25. Proposed Database Generation DDL Script (Example)

A convenient feature of the PMA application included in MAMP is the easy to navigate GUI, which allows specification and generation of desired script without having to know SQL.

From the Home screen, to which can be returned by pressing the Home Button (Figure 26), there is a quick start wizard underneath the Actions area (Figure 27).

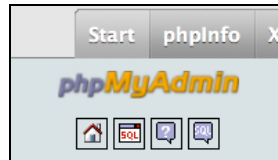


Figure 26. PMA Home Button Option

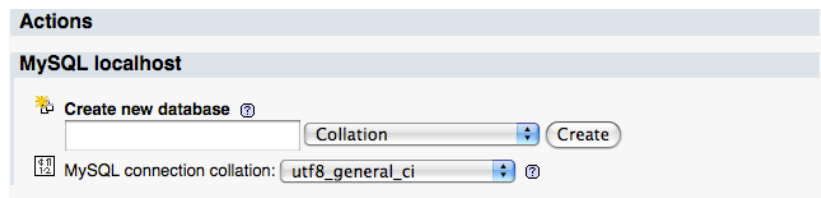


Figure 27. PMA Quick Access to Create New Database

Using this option to instantiate a database, let us call it MyDatabase, PMA would execute the following DDL:

```
CREATE DATABASE `MyDatabase` ;
```

a. Generating Tables

This step is also very straightforward. After clicking to select the desired database within which to create the table, the first tab available will be the Structure tab. This will make available an option to create a table and specify the number of fields that table should have (Figure 28). This information can be taken directly from the relational schema created in the last step. The heading of each line will be the name of the table (e.g., CRS), and each of the attributes listed will make up one of the fields, or columns, to be included. No code is executed in this step until after the details about each field are

specified in the next step. Once the code is executed, changes to the tables, like renaming, can be found under the Operations tab.

The screenshot shows the 'Structure' tab in phpMyAdmin. At the top, there are tabs for Structure, SQL, Search, Query, Export, Import, Operations, Privileges, and Drop. Below the tabs, it says 'No tables found in database.' There is a section titled 'Create new table on database MyDatabase'. It contains two input fields: 'Name:' and 'Number of fields:'. At the bottom right of this section is a 'Go' button.

Figure 28. PMA Quick Access to Create Database Table

b. Generating Fields

This part of creating the database is a bit more involved. The characteristics of the fields are based on the intended values of the record elements. A portion of this information is indicated on the relational schema. The rest is user-declared. Some basic understanding of database architecture is required to properly complete this step. An example of the PMA-provided interface can be seen in Figure 29.

The screenshot shows the 'SampleFields' table creation interface in phpMyAdmin. It has a breadcrumb trail: localhost > MyDatabase > SampleFields. The interface is divided into several sections. The top section is a table with columns: Field, Type, Length/Values, Default, Collation, Attributes, Null, Index, A.I., and Comments. There are four rows, each with a 'VARCHAR' type and various options. Below this table, there are sections for 'Table comments:', 'Storage Engine:' (set to MyISAM), 'Collation:', and 'PARTITION definition:'. At the bottom right, there is a 'Save' button, 'Or Add 1 field(s)', and a 'Go' button.

Figure 29. PMA Field Creation Options

The name of each field is placed within the Field column. The type of data contained in the field is selected from the list in the Type column. The allowable length of the data (i.e., digits or characters), or the list or set from which the user will be able to choose is set in the Length/Value column. The Default column permits the designer to set a default value when not specified by user. The Collation column specifies the character set and language. The Attribute column provides three options for how the inputted data is stored. 'BINARY' stores the string of data byte-by-byte vice character by character. This action makes spaces in the data string significant and differentiates letter case, for instance ['A' != 'a'] and ['a' != ' a']. 'UNSIGNED' restricts numeric data inputted to being positive, so all negative numbers are defaulted to '0'. 'UNSIGNED ZEROFILL' does the same as 'UNSIGNED,' however it places '0's in all value bits up to the maximum allowed length, so '-1' would be represented as '0000' if that field's length restriction is set to four digits. The Null column, if selected, permits the user specified value to be blank. If not selected, a blank input from user will yield an error message. The Index column allows the DBMS to search the database more efficiently. If specified, the DBMS will be able to locate specific column values without first having to read through the entire row of information. Very frequently data in one table references data from a different table within the database. Indexes allows for this type of referencing. The table with the indexed term offers its contents as a set from which the referencing table can choose. The A_I column represents auto increment. Selecting this column will have the DBMS to generate a

unique identification number for each row created. And last is the Comment column, which can be used to provide the user with a message (e.g., 'Example format \$12.34'). Figure 30 is an example of the type of code that PMA would execute for our proposed system's *Offerings* table.

```
CREATE TABLE `CourseOfferings` (  
  
  `Crs_No` varchar(8) NOT NULL,  
  
  `FY` int(4) NOT NULL COMMENT 'e.g. 2010',  
  
  `Qtr` set('FA1','WI2','SP3','SU4') NOT NULL,  
  
  `Sect` int(2) NOT NULL COMMENT 'e.g. 01',  
  
  `Location` varchar(7) DEFAULT NULL COMMENT 'e.g. IN263, RO200C',  
  
  PRIMARY KEY (`Crs_No`,`FY`,`Qtr`,`Sect`)  
  
  ) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Figure 30. Proposed DBMS *Offerings* Table DDL

B. MANAGING THE DATABASE

1. Data Manipulation Language (DML)

DML is the DBMS scripting language used to manipulate, or perform functions on, the data within the database. Some typical functions include retrieval, insertion, deletion, and modification of the data. Some common commands used to carry out these operations are: SELECT, UPDATE, INSERT, & DELETE; which are very easily associated with the purposes they fulfill within DML.

The two main types of DML are high-level and low-level DML. High-level DML is also known as nonprocedural DML. Languages considered to be high-level DML are declarative languages and set-oriented. They declare what data is to be retrieved vice how or what procedures by which that data is to be retrieved. These languages can be utilized by

themselves or embedded within a general-purpose programming language like C++ or Java. SQL, a relational language, is one type of High-level DML.

Low-level, or procedural, DML is retrieves data one record at a time vice as a set. Language of this type must be embedded within general-purpose programming languages (Elmasri & Navathe, 2007).

2. Adding Records

To add records within the database, PMA offers an Insert tab option. This is visible after selecting a particular table to modify (Figure 31). The interface provided utilizes dropdown lists to facilitate limited choice options and free-select options to facilitate attributes with the possibility of multiple values. All other field input areas are set based on specification in the DDL Script executed. Notice, in Figure 31, *FacName* also has a dotted underlining. This indicates a comment, which appears when the mouse hovers over that field name (Figure 32).

Field	Type	Function	Null	Value
<u>FacName</u>	varchar(40)			
Acad_Rank	enum	--	<input checked="" type="checkbox"/>	
Tenure_Trk	enum	--	<input checked="" type="checkbox"/>	
Op_Status	enum	--		Active
Qtr_Prefs	set	--		FA1 W12 SP3 SU4

Go

Figure 31. PMA Insert Tab for Record Adding

Field	Type	Function	Null
FacName	varchar(40)		
Acad_Rank	enum	--	<input checked="" type="checkbox"/>

Figure 32. PMA Comment (Insert Tab)

After entering record data, as shown in Figure 33, PMA will generate the appropriate SQL script to implement the addition. An example of this INSERT script for the *Faculty* table of the proposed database is shown in Figure 34.

Field	Type	Function	Null	Value
FacName	varchar(40)			Doe, John
Acad_Rank	enum	--	<input type="checkbox"/>	Professor
Tenure_Trk	enum	--	<input type="checkbox"/>	-- Tenured
Op_Status	enum	--		Active
Qtr_Prefs	set	--		FA1 WI2 SP3 SU4

Go

Figure 33. PMA Insert Tab (Completed Fields)

```
INSERT INTO `ThesisDB_3b`.`FacultyMembers` (
  `FacName`,
  `Acad_Rank`,
  `Tenure_Trk`,
  `Op_Status`,
  `Qtr_Prefs`
)
VALUES (
  'Griffith, Peter', 'Professor', 'Tenured', 'Active', 'WI2,SU4'
);
```

Figure 34. INSERT *Faculty* Table DDL Script Example

3. Editing Records

Making modifications to a record can be done by selecting the table in which the record resides, then by choosing the Browse tab. This action will display all of

the records within that table. Next to each record is shown a Selection box, Pencil tool (to edit) and a red X (to delete) to corresponding record. Each of these tools can be seen in Figure 35 within the aforementioned Browse tab.

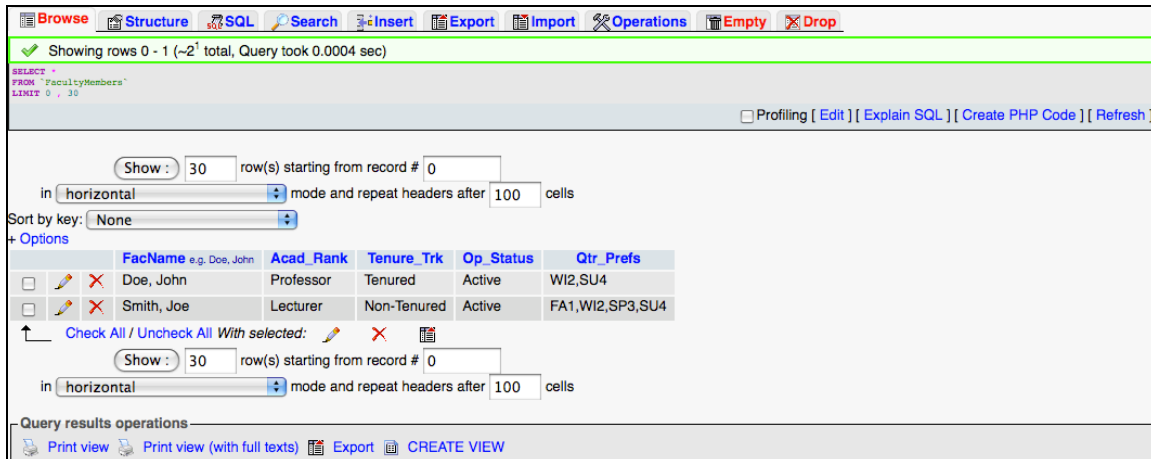


Figure 35. PMA Browse Tab Options

Selecting the Pencil tool will redirect to the Insert tab with that record's fields pre-completed. After making the desired modifications, pressing Go will execute the Update script appropriate to execute the changes within the database. Figure 36 depicts an example of the DDL script executed.

```
UPDATE `ThesisDB_3b`.`FacultyMembers`
SET `FacName` = 'Smith, Joseph'
WHERE `FacultyMembers`.`FacName` = 'Smith, Joe';
```

Figure 36. Faculty Table UPDATE DDL Script Example

4. Deleting Records

Deleting records is done in much the same fashion as updating or modifying records with an exception to the last

step. From the Browse tab, instead of selecting the Pencil tool, select the red X. Next confirm DELETE execution (Figure 37) and the records will be redisplayed minus that record.

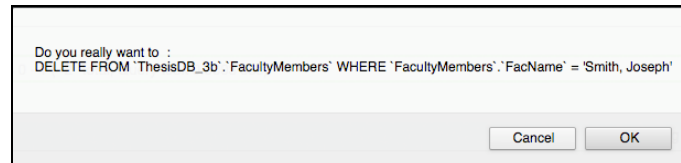


Figure 37. DELETE DDL Script Example

5. Importing Records

The import feature can be a very useful option in transitioning in-between any database, especially as content increases in quantity. MySQL makes this option available through variations of the LOAD DATA INFILE syntax as seen in (Figure 38).

```
LOAD DATA [LOW_PRIORITY | CONCURRENT]
[LOCAL] INFILE 'file_name'
[REPLACE | IGNORE]
INTO TABLE tbl_name
[CHARACTER SET charset_name]
[({FIELDS | COLUMNS}
  [TERMINATED BY 'string']
  [[OPTIONALLY] ENCLOSED BY 'char']
  [ESCAPED BY 'char']
)]
```

Figure 38. LOAD DATA DDL Syntax Template (From MySQL, 2011)

PMA offers a more user friendly solution with a GUI available under the Import Tab (Figure 39). Several document types are accepted for import including CSV files, which can easily be generated directly from Excel files.

File to Import:

File may be compressed (gzip, bzip2, zip) or uncompressed.
A compressed file's name must end in `[format].[compression]`. Example: `.sql.zip`

Browse your computer: (Max: 32MiB)

Character set of the file:

Partial Import:

☒ Allow the interruption of an import in case the script detects it is close to the PHP timeout limit. *(This might be a good way to import large files, however it can break transactions.)*

Number of rows to skip, starting from the first row:

Format:

Format-Specific Options:

SQL compatibility mode:

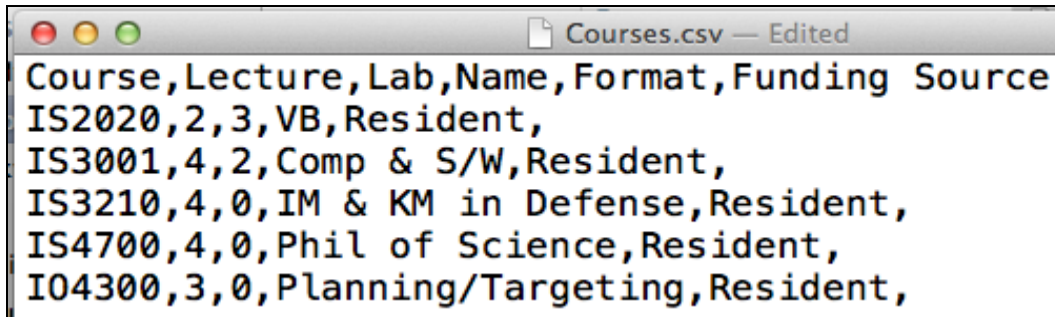
☒ Do not use `AUTO_INCREMENT` for zero values

Figure 39. Sample of PMA Import Tab Wizard

Simple structure importing is rather straightforward. Figure 40, Figure 41, and Figure 42 show simplified examples of our original database represented as an Excel file, CSV file, and within PMA after importing to our proposed DBMS, respectively.

Course	Lecture	Lab	Name	Format	Funding Source
IS2020		2	3 VB	Resident	
IS3001		4	2 Comp & S/W	Resident	
IS3210		4	0 IM & KM in Defense	Resident	
IS4700		4	0 Phil of Science	Resident	
IO4300		3	0 Planning/Targeting	Resident	

Figure 40. Original Database Sample (Courses.xls)

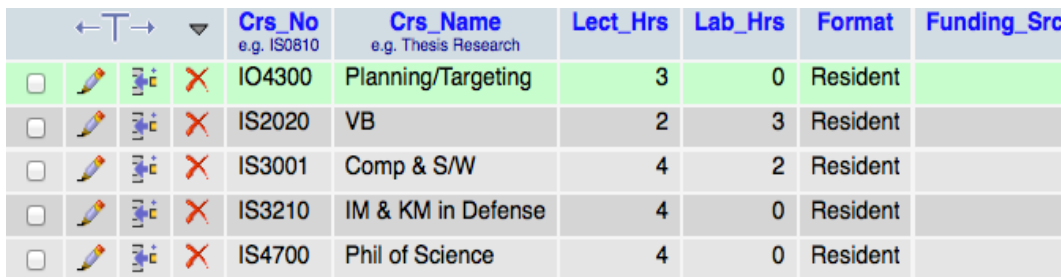


```

Course,Lecture,Lab,Name,Format,Funding Source
IS2020,2,3,VB,Resident,
IS3001,4,2,Comp & S/W,Resident,
IS3210,4,0,IM & KM in Defense,Resident,
IS4700,4,0,Phil of Science,Resident,
IO4300,3,0,Planning/Targeting,Resident,

```

Figure 41. Original Database Sample (Courses.csv)



				Crs_No e.g. IS0810	Crs_Name e.g. Thesis Research	Lect_Hrs	Lab_Hrs	Format	Funding_Src
<input type="checkbox"/>				IO4300	Planning/Targeting	3	0	Resident	
<input type="checkbox"/>				IS2020	VB	2	3	Resident	
<input type="checkbox"/>				IS3001	Comp & S/W	4	2	Resident	
<input type="checkbox"/>				IS3210	IM & KM in Defense	4	0	Resident	
<input type="checkbox"/>				IS4700	Phil of Science	4	0	Resident	

Figure 42. Proposed Database Sample Post-Import Courses.csv

The complexity of importing becomes more apparent when attempting to import data into a database with entities (tables) enforcing foreign key constraints. The delicacy of this operation is due to strict adherence to accuracy demanded of most programming languages.

When importing data under the conditions aforementioned, the user must make sure that data content within the original database meets the criteria of the new database. For example, if the *Course* attribute *Lecture Hours* within the DBMS is expecting an input of integers only, importing a data instance of "2" represented as the string "two" or character "2" will cause an error upon executing the code. Equally, if data from the original database is imported into a field (column) that draws its inputs from a limited set of data, an error will result if the imported data does not match an item from that set.

This also implies that the order of importing matters. If the set of data to be referenced within a field has not yet been populated, importing data into the dependent field will yield an error.

Figure 43 is the code that PMA executes in order to import the data from the CSV file in Figure 41 to the DBMS in Figure 42. This is an alternative code to using the LOAD DATA version. Figure 44 depicts PMA's LOAD DATA INFILE code which is also an option under PMA's Import Tab.

```
INSERT INTO `Curriculums`  
VALUES (  
    'IS2020', 'IS', 'Required'  
)# 1 row affected.  
  
INSERT INTO `Curriculums`  
VALUES (  
    'IS3001', 'IS', 'Elective'  
)# 1 row affected.  
  
INSERT INTO `Curriculums`  
VALUES (  
    'IS3210', 'IS', 'Elective'  
)# 1 row affected.  
  
INSERT INTO `Curriculums`  
VALUES (  
    'IS4700', 'IS PHD', 'Required'  
)# 1 row affected.  
  
INSERT INTO `Curriculums`  
VALUES (  
    'IO4300', 'IO/IW', 'Required'  
)# 1 row affected.
```

Figure 43. PMA Import DDL Script Using INSERT Function

```

LOAD DATA LOCAL INFILE '/Applications/MAMP/tmp/php/phpLxkCtl'
INTO TABLE `CRS`
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
ESCAPED BY '\\'
LINES TERMINATED BY '\n'
IGNORE 1 LINES (
  `Crs_No`,
  `Lect_Hrs`,
  `Lab_Hrs`,
  `Crs_Name`,
  `Format`,
  `Funding_Src`
)

```

Figure 44. PMA Import DDL Script Using LOAD DATA Function

C. VIEWS & REPORTS

1. Creating Views

Views offer a customized perspective, or view, of the content contained within the same database for multiple users with differing requirements. These views can show data from a single table; present compiled data from several tables; or even display derived data, not explicitly stored in the database, from multiple instances of data that are explicitly stored in the database (Elmasri & Navathe, 2007). For example, a view could be designed to display 'Costs 36.00' derived from *Tenure_Trk.Cost* of '12.00' with three instances in the *Offerings* Table. This would utilize the multiplication function built into the MySQL's DML. This is just one of many including absolute value (ABS), pi (PI), radians (RADIANS), sine (SIN), natural logarithm (LN), etc. (MySQL, 2011).

Most operations within PMA have a relatively simple interface that allows users & designers to interact with the database and DBMS. The view function exists as an option at the bottom of a selected table's Browse Tab (Figure 45). However, unlike most other PMA interfaces, PMA's create view option requires the designer to be more

than just a little familiar with MySQL syntax, especially regarding CREATE VIEW DDL. Figure 46 shows the layout of PMA's CREATE VIEW screen and Figure 47 shows the CREATE VIEW Template provided by MySQL (MySQL, 2011).

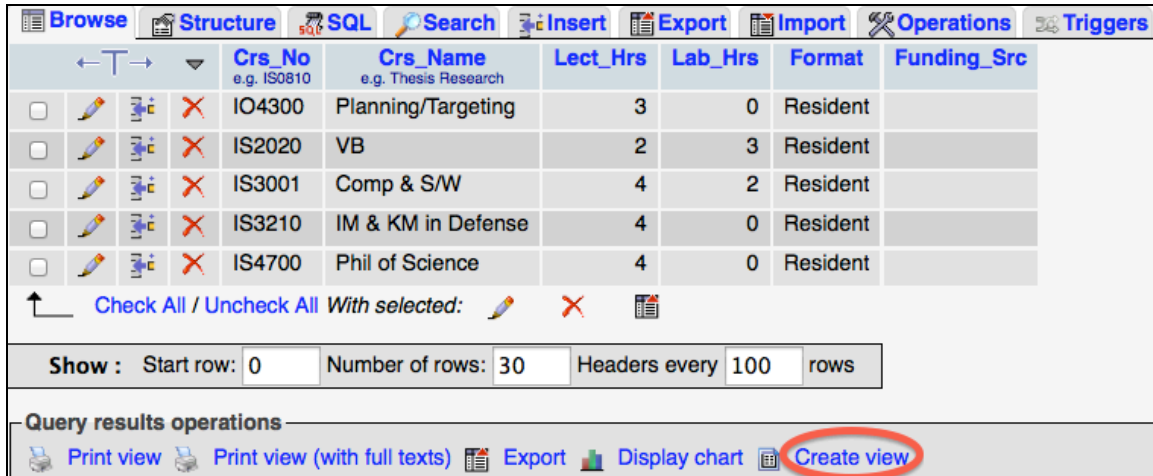


Figure 45. PMA Create View Option

One shortcut to designing the desired view is through generation of a query. To create a query, select the database of interest and go to the Query tab (Figure 48). The Column dropdown menu will allow the user to select from the range attribute fields, or columns, within the database. The sort option is pretty self-explanatory. Show allows the use of a column to set criteria while making it optional for the column to display in the query results. Criteria boxes allow the user to specify a type of filtering option for what is displayed in the query results. The SQL syntax used in this input field is very sensitive to accuracy and SQL protocol in accordance with the user's version of MySQL. For instance, when specifying the *Cost* field within the *Tenure_Trk* table, the user must take care to use the prime symbol (```), which can be found

above the Tab Button on the keyboard's Tilde key vice the apostrophe (`); the two of which can be easily mistaken. This will result in SQL code like: `Tenure_Trk`.`Cost`='36.00', where the prime symbol encloses field and table names and the apostrophe encloses string characters. The first set of 'Ins, Del, And, Or' clusters underneath Criteria allow the addition or subtraction of And or Or Statements to a column. The second set along the Modify Row allow the addition or subtraction of And or Or between query columns. Columns and rows can also be added and deleted using the dropdown menus below the aforementioned set of query options (Figure 48).

After generating the query with the desired data displayed, simply press the Create view option below the query results and the appropriate code will be generated in order to create the respective view (Figure 46).

CREATE VIEW

OR REPLACE ☐

ALGORITHM **UNDEFINED** ▾

VIEW name

Column names

AS

```
SELECT * FROM `CRS`
```

WITH ☐ CASCADED CHECK OPTION ☐ LOCAL CHECK OPTION

Figure 46. PMA Create View Screen Option

```

CREATE
    [OR REPLACE]
    [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
    [DEFINER = { user | CURRENT_USER }]
    [SQL SECURITY { DEFINER | INVOKER }]
    VIEW view_name [(column_list)]
    AS select_statement
    [WITH [CASCADED | LOCAL] CHECK OPTION]

```

Figure 47. CREATE VIEW Syntax Template

Figure 48. PMA Query Tab

2. Reports

Essentially, views can act as reports generated from the database. They are designed to be customized to show the data a user would like to see and they are saved along with tables in the DBMS (Figure 49).













	Table	Action	Rows	Type	Collation	Size	Overhead
<input checked="" type="checkbox"/>	CourseView	     	5	View	---	-	-
<input type="checkbox"/>	CRS	     	5	InnoDB	latin1_swedish_ci	16 KiB	-

Figure 49. PMA View Placement (Example)

Once the views are designed as preferred, there is also an export feature available within a selected view.

Although, not the most robust report-generating option, it does provide a neatly assembled output in several file formats including Portable Document Format (PDF) and CSV.

THIS PAGE INTENTIONALLY LEFT BLANK

V. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSION

1. Solution

The proposed database and its re-engineered DBMS addressed the problems associated with the original database. It adds a great deal more concurrency control and streamlined usage.

The proposed database is fairly easy to duplicate. Development takes a bit of time, which is reduced with experience using PMA. Most of the time associated with this development is rooted in quality assurance. As with any form of computer programming, the developer must check and check again to verify mistakes are minimized and general user operation will be as smooth as possible. It took approximately 20-40 hours to design the database from conceptual understanding of desired database through development of the relational schema. Generating the DDL Script and programming required about half the time, not accounting for fine-tuning and bug fixing.

Many reiterations were required to truly reach the desired database. Since the methodology followed a cyclical re-engineering process, many more iterations are expected. These reiterations are no different than typical software updates and programs changes in order to keep the application relevant.

As stated in the assumptions, or general expectations, the proposed solution was to explore available scheduling tools operational on the Macintosh Operating System with

the smallest practical price tag. The solution was also to be quickly implementable and remotely web-enabled. Each of the aforementioned elements of functionality were accommodated.

B. RECOMMENDATIONS

1. Future Application

SQL is a language made to manage data in databases. Databases are prevalent in maintaining order in almost every walk of civilization from banking, education, entertainment. Most industries of any type rely on databases and subsequently create a market for SQL.

MySQL is the self-proclaimed "world's most popular open source database." Regardless, if not, it only seems to be rivaled by one other, PostgreSQL. Their continued use is a testament to their powerful design outside of their freeware price. Also, since they are open source, third parties are free to modify them as they see fit. SQL has even proven itself good enough for government work with use by Defense Security Cooperation Assistance amongst others (What We Do, n.d.).

2. Additional Research

Aside from the expected iterations to maintain the database, additional research on report generations is necessary. Report generating was broadly covered within the scope of this thesis; however the products are not very robust and customizable in appearance. There are a number of relatively standard outputs PMA can generate via its export feature which may naturally lend themselves to adaptations with PDF or Microsoft Word.

There is more to look into regarding MAMP's use of "the cloud," which references the data storage universally accessible on the internet. In this thesis project, inputs files located on "the cloud" were utilized but it is unknown whether the database and its contents being managed by PMA can be actively located on "the cloud."

Lastly, this particular database was being maintained on a DoD computer. Since all the content was locally maintained on the computer, the security aspect was not much of a concern. However, since the DBMS is also design to be web-capable, or remotely accessible via internet, the question still remains whether web access can be established securely in accordance with the government VPN protocols. Delving more deeply into the native security features of PMA and ascertaining whether web use is plausible under the aforementioned conditions would open a new line of possibilities in mobile and home database access.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX

This is the actual MySQL code generated by PMA when the desired database specifications are entered. Some changes were made after several iterations of the re-engineering process discussed throughout this thesis.

```
--
-- Database: `IS Dept DB`
-- DDL for Actual Proposed Database (Reformatted for structure illustration)
-- Creator: Reed, Gary
-- Creation Date: January 10, 2011
--
CREATE DATABASE `IS Dept DB` DEFAULT CHARACTER SET latin1 COLLATE latin1_swedish_ci;
USE `IS Dept DB`;

-----

--
-- Stand-in structure for view `2011 Course Offerings (by Curriculum then Quarter)`
--
CREATE TABLE `2011 Course Offerings (by Curriculum then Quarter)` (
  `Curriculum` enum('IS','Elective','IS PhD','IO/IW','CC','DL','HLS'),
  `Course` varchar(8),
  `Lecture` int(2),
  `Lab` int(2),
  `Name` varchar(40),
  `Quarter` enum('FA1','WI2','SP3','SU4'),
  `Instructor` varchar(40),
  `# of Sections` int(2)
);

-----

--
-- Stand-in structure for view `2011 Course Offerings (by course no)`
--
CREATE TABLE `2011 Course Offerings (by course no)` (
  `Courses` varchar(8),
  `Lecture` int(2),
  `Lab` int(2),
  `Name` varchar(40),
  `Quarter` enum('FA1','WI2','SP3','SU4'),
  `Instructor` varchar(40),
  `# of Offerings` int(2)
);

-----

--
-- Stand-in structure for view `2011 Offered Section Count`
--
CREATE TABLE `2011 Offered Section Count` (
  `Track` enum('Tenure','Non-Tenure','Military'),
  `Instructor` varchar(40),
  `# of Sections` decimal(32,0)
);

-----

--
-- Stand-in structure for view `2011 Section Costs (by Track)`
--
CREATE TABLE `2011 Section Costs (by Track)` (
  `Track` enum('Tenure','Non-Tenure','Military'),
  `Total Sections` decimal(54,0),
  `Total Cost` decimal(60,2)
);

-----

--
-- Table structure for table `COST`
```

```

--
CREATE TABLE `COST` (
  `Track` enum('Tenure','Non-Tenure','Military','') NOT NULL,
  `Price (per section)` decimal(6,2) DEFAULT NULL COMMENT 'e.g. 12.34',
  KEY `Track` (`Track`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

-----

--
-- Table structure for table `COURSES`
--

CREATE TABLE `COURSES` (
  `CourseNumber` varchar(8) NOT NULL COMMENT 'e.g. IS0810',
  `LectureHours` int(2) DEFAULT NULL COMMENT 'e.g. 2',
  `LabHours` int(2) DEFAULT NULL COMMENT 'e.g. 2',
  `CourseName` varchar(40) NOT NULL COMMENT 'e.g. Thesis Research',
  `Coordinator` varchar(40) DEFAULT NULL COMMENT 'e.g. Doe, John; Doe',
  `AssociatedCurriculum(s)` enum('IS','Elective','IS PhD','IO/IW','CC','DL','HLS') NOT NULL,

  PRIMARY KEY (`CourseNumber`,`AssociatedCurriculum(s)`),
  KEY `Coordinator` (`Coordinator`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

-----

--
-- Table structure for table `FACULTY`
--

CREATE TABLE `FACULTY` (
  `FacultyName` varchar(40) NOT NULL COMMENT 'e.g. Doe, John; Doe',
  `OperationalStatus` enum('Active','Inactive') DEFAULT NULL,
  `AcademicRank` enum('Professor','Assistant Professor','Research Professor','Senior Lecturer','Lecturer','Military') DEFAULT
NULL,
  `TenureTrack` enum('Tenure','Non-Tenure','Military') DEFAULT NULL,
  `QuarterPreferences` set('FA1','WI2','SP3','SU4') DEFAULT NULL,
  `AdditionalNotes` varchar(256) DEFAULT NULL,

  PRIMARY KEY (`FacultyName`),
  KEY `TenureTrack` (`TenureTrack`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

-----

--
-- Table structure for table `LOCATIONS`
--

CREATE TABLE `LOCATIONS` (
  `CourseNumber` varchar(8) NOT NULL COMMENT 'e.g. IS0810',
  `SectionNumber` int(2) NOT NULL DEFAULT '0' COMMENT 'e.g. 01',
  `Location` varchar(7) NOT NULL COMMENT 'e.g. RO202C',

  PRIMARY KEY (`CourseNumber`,`SectionNumber`),
  KEY `CourseNumber` (`CourseNumber`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

-----

--
-- Table structure for table `OFFERINGS`
--

CREATE TABLE `OFFERINGS` (
  `CourseNumber` varchar(8) NOT NULL COMMENT 'e.g. IS0810',
  `FiscalYear` int(4) NOT NULL COMMENT 'e.g. 2010',
  `Quarter` enum('FA1','WI2','SP3','SU4') NOT NULL,
  `Instructor` varchar(40) NOT NULL DEFAULT '' COMMENT 'e.g. Doe, John; Doe',
  `SectionAmount` int(2) DEFAULT NULL COMMENT 'e.g. 2, i.e. # of sections instructor offers',

  PRIMARY KEY (`CourseNumber`,`FiscalYear`,`Quarter`,`Instructor`),
  KEY `CourseNumber` (`CourseNumber`,`Instructor`),
  KEY `Instructor` (`Instructor`),
  KEY `FiscalYear` (`FiscalYear`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

```

```

-----
--
-- Table structure for table `QUALIFICATIONS`
--

CREATE TABLE `QUALIFICATIONS` (
  `FacultyName` varchar(40) NOT NULL COMMENT 'e.g. Doe, John; Doe',
  `CourseNumber` varchar(8) NOT NULL COMMENT 'e.g. IS0810',

  PRIMARY KEY (`FacultyName`, `CourseNumber`),
  KEY `CourseNumber` (`CourseNumber`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

-----

--
-- Structure for view `2011 Course Offerings (by Curriculum then Quarter)`
--
DROP TABLE IF EXISTS `2011 Course Offerings (by Curriculum then Quarter)`;

CREATE ALGORITHM=UNDEFINED VIEW `2011 Course Offerings (by Curriculum then Quarter)` AS select
  `COURSES`.`AssociatedCurriculum(s)` AS `Curriculum`,
  `COURSES`.`CourseNumber` AS `Course`,
  `COURSES`.`LectureHours` AS `Lecture`,
  `COURSES`.`LabHours` AS `Lab`,
  `COURSES`.`CourseName` AS `Name`,
  `OFFERINGS`.`Quarter` AS `Quarter`,
  `OFFERINGS`.`Instructor` AS `Instructor`,
  `OFFERINGS`.`SectionAmount` AS `# of Sections`

from (
  `COURSES` join `OFFERINGS`
)

where ((
  `OFFERINGS`.`CourseNumber` = `COURSES`.`CourseNumber`
)
and (
  `OFFERINGS`.`CourseNumber` = `COURSES`.`CourseNumber`
)
and (
  `OFFERINGS`.`FiscalYear` = '2011'
))

order by
  `COURSES`.`AssociatedCurriculum(s)`,
  `OFFERINGS`.`Quarter`;

-----

--
-- Structure for view `2011 Course Offerings (by course no)`
--
DROP TABLE IF EXISTS `2011 Course Offerings (by course no)`;

CREATE ALGORITHM=UNDEFINED VIEW `2011 Course Offerings (by course no)` AS
select
  `COURSES`.`CourseNumber` AS `Courses`,
  `COURSES`.`LectureHours` AS `Lecture`,
  `COURSES`.`LabHours` AS `Lab`,
  `COURSES`.`CourseName` AS `Name`,
  `OFFERINGS`.`Quarter` AS `Quarter`,
  `OFFERINGS`.`Instructor` AS `Instructor`,
  `OFFERINGS`.`SectionAmount` AS `# of Offerings`

from (
  `COURSES` join `OFFERINGS`
)

where ((
  `OFFERINGS`.`CourseNumber` = `COURSES`.`CourseNumber`
)
and (
  `OFFERINGS`.`CourseNumber` = `COURSES`.`CourseNumber`
)
and (

```

```

        `OFFERINGS`.`FiscalYear` = '2011'
    ));

-----

--
-- Structure for view `2011 Offered Section Count`
--
DROP TABLE IF EXISTS `2011 Offered Section Count`;

CREATE ALGORITHM=UNDEFINED VIEW `2011 Offered Section Count` AS
    select
        `FACULTY`.`TenureTrack` AS `Track`,
        `2011 Course Offerings (by Curriculum then Quarter)`.`Instructor` AS `Instructor`,
        sum(
            `2011 Course Offerings (by Curriculum then Quarter)`.`# of Sections`
        ) AS `# of Sections`
    from (
        `2011 Course Offerings (by Curriculum then Quarter)` join `FACULTY`
    )
    where (
        `FACULTY`.`FacultyName` = `2011 Course Offerings (by Curriculum then Quarter)`.`Instructor`
    )
    group by
        `2011 Course Offerings (by Curriculum then Quarter)`.`Instructor`
    order by
        `FACULTY`.`TenureTrack`, `2011 Course Offerings (by Curriculum then Quarter)`.`Instructor`;

-----

--
-- Structure for view `2011 Section Costs (by Track)`
--
DROP TABLE IF EXISTS `2011 Section Costs (by Track)`;

CREATE ALGORITHM=UNDEFINED VIEW `2011 Section Costs (by Track)` AS
    select
        `2011 Offered Section Count`.`Track` AS `Track`,
        sum(
            `2011 Offered Section Count`.`# of Sections`
        ) AS `Total Sections`,
        (`COST`.`Price (per section)` * sum(`2011 Offered Section Count`.`# of Sections`))
        AS `Total Cost`
    from (
        `2011 Offered Section Count` join `COST`
    )
    where (
        `COST`.`Track` = `2011 Offered Section Count`.`Track`
    )
    group by
        `2011 Offered Section Count`.`Track`;

--
-- Constraints for dumped tables
--

--
-- Constraints for table `COST`
--
ALTER TABLE `COST`
  ADD CONSTRAINT `COST_ibfk_2` FOREIGN KEY (`Track`) REFERENCES `FACULTY` (`TenureTrack`)
  ON DELETE NO ACTION ON UPDATE CASCADE,
  ADD CONSTRAINT `COST_ibfk_1` FOREIGN KEY (`Track`) REFERENCES `FACULTY` (`TenureTrack`)
  ON DELETE NO ACTION ON UPDATE CASCADE;

--
-- Constraints for table `COURSES`
--

```

```

ALTER TABLE `COURSES`
  ADD CONSTRAINT `COURSES_ibfk_2` FOREIGN KEY (`Coordinator`) REFERENCES `FACULTY` (`FacultyName`)
    ON UPDATE CASCADE,
  ADD CONSTRAINT `COURSES_ibfk_1` FOREIGN KEY (`Coordinator`) REFERENCES `FACULTY` (`FacultyName`)
    ON UPDATE CASCADE;

--
-- Constraints for table `LOCATIONS`
--
ALTER TABLE `LOCATIONS`
  ADD CONSTRAINT `LOCATIONS_ibfk_2` FOREIGN KEY (`CourseNumber`) REFERENCES `OFFERINGS` (`CourseNumber`)
    ON DELETE CASCADE ON UPDATE CASCADE,
  ADD CONSTRAINT `LOCATIONS_ibfk_1` FOREIGN KEY (`CourseNumber`) REFERENCES `OFFERINGS` (`CourseNumber`)
    ON DELETE CASCADE ON UPDATE CASCADE;

--
-- Constraints for table `OFFERINGS`
--
ALTER TABLE `OFFERINGS`
  ADD CONSTRAINT `OFFERINGS_ibfk_3` FOREIGN KEY (`CourseNumber`) REFERENCES `COURSES` (`CourseNumber`)
    ON DELETE CASCADE ON UPDATE CASCADE,
  ADD CONSTRAINT `OFFERINGS_ibfk_4` FOREIGN KEY (`Instructor`) REFERENCES `FACULTY` (`FacultyName`)
    ON UPDATE CASCADE,
  ADD CONSTRAINT `OFFERINGS_ibfk_1` FOREIGN KEY (`CourseNumber`) REFERENCES `COURSES` (`CourseNumber`)
    ON DELETE CASCADE ON UPDATE CASCADE,
  ADD CONSTRAINT `OFFERINGS_ibfk_2` FOREIGN KEY (`Instructor`) REFERENCES `FACULTY` (`FacultyName`)
    ON UPDATE CASCADE;

--
-- Constraints for table `QUALIFICATIONS`
--
ALTER TABLE `QUALIFICATIONS`
  ADD CONSTRAINT `QUALIFICATIONS_ibfk_3` FOREIGN KEY (`FacultyName`) REFERENCES `FACULTY` (`FacultyName`)
    ON DELETE NO ACTION ON UPDATE CASCADE,
  ADD CONSTRAINT `QUALIFICATIONS_ibfk_4` FOREIGN KEY (`CourseNumber`) REFERENCES `COURSES` (`CourseNumber`)
    ON DELETE NO ACTION ON UPDATE CASCADE,
  ADD CONSTRAINT `QUALIFICATIONS_ibfk_1` FOREIGN KEY (`FacultyName`) REFERENCES `FACULTY` (`FacultyName`)
    ON DELETE NO ACTION ON UPDATE CASCADE,
  ADD CONSTRAINT `QUALIFICATIONS_ibfk_2` FOREIGN KEY (`CourseNumber`) REFERENCES `COURSES` (`CourseNumber`)
    ON DELETE NO ACTION ON UPDATE CASCADE;

```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

- Ahmed, M., Uddin, M. M., Azad, M. S., & Haseeb, S. (2010). MySQL performance analysis on a limited resource server: Fedora vs. ubuntu linux. *Proceedings of the 2010 Spring Simulation Multiconference*, 99, 1-7.
- Alavi, M., & Leidner, D. (2001). Review: Knowledge management and knowledge management systems: Conceptual foundations and research issues. *MIS Quarterly*, 25 (1), 107-136.
- Ambler, S. (2002-2011). *Data Modeling 101* (Ambysoft). Retrieved from <http://www.agiledata.org/essays/dataModeling101.html>
- Dairymple, J. (2003, June 13). *Macworld*. Retrieved February 2011, from <http://www.macworld.com/article/24898/2003/06/explorer.html>
- Densham, P. (1991). Spatial decision support systems. In M. Goodchild, D. Maguire, M. Goodchild, & D. Rhind (Eds.), *Geographical information systems: Principles and applications* (vol. 2, pp. 403-412). London: Longman.
- Dhesi, R. (2011). *Prevalent Database Models*. Retrieved March 2011, from <http://www.randipdhesi.com/project/images/fig1-2.jpg>
- Elam, J., & Leidner, D. (1993). Executive information systems: their impact on executive decision making. *Journal of Management Information Systems-Special issue: Organizational impact of group support systems, expert systems, and executive information systems*, 10 (3), 139-155.
- Elmasri, R., & Navathe, S. (2007). *Fundamentals of database systems* (5th ed.). Boston, MA: Pearson & Addison Wesley.
- Eom, S. (2001). Decision support systems. *International Encyclopedia of Business and Management* (2). (M. Warner, Ed.) London: International Thomson Business Publishing.

- Harlan, E. (Ed.). (2009, October 8). Pages: *Creating business processes in SharePoint*. Retrieved from <http://www.baltimoresug.org/Resources/videos/Pages/BSUG%20Introduction%20to%20Workflow%20Process%20Modeling%20in%20SharePoint.pptx>
- Hättenschwiler, P. (1999). Neues anwenderfreundliches konzept der entscheidungsunterstützung. Gutes entscheiden in wirtschaft, politik und gesellschaft. Zurich, vdf Hochschulverlag AG: 189-208.
- Import Excel data into MySQL in 5 easy steps*. (2011). Retrieved February 2011, from <http://blog.tjitjing.com/index.php/2008/02/>
- Kambalyal, C. (n.d.). *3-Tier Architecture*. Retrieved March 2011, from <http://channukambalyal.tripod.com/NTierArchitecture.pdf>
- Keen, P., & Scott-Morton, M. (1978). *Decision support systems: An organizational perspective*. Reading, MA: Addison-Wesley Publishing Company.
- Kennedy, M. (2010, August). Evaluating open source software. *Defense AT&L*, 42-45.
- Kim, W. (1990). Object-oriented databases: Definition and research directions. *Knowledge and Data Engineering, IEEE Transactions on*, 2 (3), 327-341.
- Lai, A., & Nieh, J. (2006). On the performance of wide-area thin-client computing. *ACM Transactions on Computer Systems*, 24 (2), 175-209.
- MapsofIndia.com. (2009). Attribute data models. Retrieved March 2011, from <http://www.mapsofindia.com/images/network-model.jpg>
- Minoli, D. (2008). *Enterprise architecture A to Z: frameworks, business process modeling, soa, and infrastructure technology*. Boca Raton, FL: Auerbach Publications.
- MySQL. (2011). *MySQL 5.1 reference manual*. Retrieved February 2011, from <http://dev.mysql.com/doc/refman/5.1/en/load-data.html>

- Netcraft. (2009, February 18). *February 2009 web server survey*. Retrieved February 2011, from http://news.netcraft.com/archives/2009/02/18/february_2009_web_server_survey.html
- Netcraft. (2011). *Web server survey: Top servers across all domains*. Retrieved 2011, from <http://news.netcraft.com/archives/category/web-server-survey/>
- Nordbotten, J., & Crosby, M. (1999, July). *An experiment in data model perception*. Retrieved March 2011, from <http://nordbotten.com/joan/dmp/Oodm-prj.jpg>
- Object oriented databases*. (2010, March 27). Retrieved March 2011, from <http://www.comptechdoc.org/independent/database/basicdb/dataobject.html>
- Oracle. (2012). 13.6.4.4. FOREIGN KEY constraints. Retrieved from <http://dev.mysql.com/doc/refman/5.1/en/innodb-foreign-key-constraints.html>
- Oracle. (2013). 13.2.6. LOAD DATA infile syntax. Retrieved from <http://dev.mysql.com/doc/refman/5.1/en/load-data.html>
- The PHP Group. (2001-2011). *What is php?*. Retrieved from <http://php.net/>
- Power, D. (1997). What is a DSS? *The On-Line Executive Journal for Data-Intensive Decision Support*, 1(3).
- Power, D. (2002). *Decision support systems: Concepts and resources for managers*. Westport, CT: Quorum Books.
- Reich, Y., & Kapeliuk, A. (2005). A framework for organizing the space of decision problems with application to solving subjective, context-dependent problems. *Decision Support Systems*, 41(1), 1-19.
- Shim, J., Warkentin, M., Courtney, J., Power, D., Sharda, R., & Carlsson, C. (2002). Past, present, and future of decision support technology. *Decision Support Systems*, 33(2), 111-126.

- Trustees of Indiana University. (2006, April 24). What are flat file and relational databases? Retrieved March 2011, from <http://kb.iu.edu/data/ahrp.html>
- W3Counter. (2011, February). W3Counter. Retrieved February 2011, from <http://www.w3counter.com/globalstats.php>
- Whatever happened to object-oriented databases? (2011). Retrieved 2011, from http://www.leavcom.com/db_08_00.htm
- What we do. (n.d.). Retrieved 2011-2013, from www.dsca.mil/jobs/positions.htm
- Wikipedia. (2011, March). *Flat file database*. Retrieved March 2011, from http://en.wikipedia.org/wiki/Flat_file_database
- Zak, N. (2008, January 7). Hierarchical data and scope checking in detail. Retrieved March 2011, from <http://www.nolanzak.com/whitepapers/HierarchicalData/nzHier1.JPG>
- Zeleny, M. (1987). Management support systems: Towards integrated knowledge management. *Human Systems Management*, 7(1), 59-70.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
Ft. Belvoir, Virginia
2. Dudley Knox Library
Naval Postgraduate School
Monterey, California